CANN HCCL通信库介绍与分享

肖士忠、颜业峰、文学敏
2025.12.04

CANN

目录

HCCL是什么：CANN的集合通信库，为昇腾计算集群提供高性能集合通信服务

HCCL（Huawei Collective Communication Library）

华为集合通信库，是CANN的基础组件之一，依托昇腾芯片高效的通信引擎与总线/网络协议，为昇腾计算集群提供**高性能、高可靠、高易用集合通信**解决方案。

高性能通信算法

- 拓扑感知的高性能通信算法，为不同业务场景、不同拓扑架构下的计算集群提供高效通信方案。

独立通信加速引擎

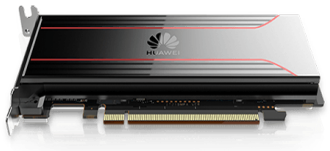
- 独立通信引擎，支持随路计算，不占用计算核，支持计算与通信高效并发。
- 硬化调度和通信原语，降低调度开销，精准控制系统抖动，提升计算集群算力利用率

https://gitcode.com/cann

CANN

计算算力发展：大算力需要分布式集群计算，且规模越来越大

AI识别

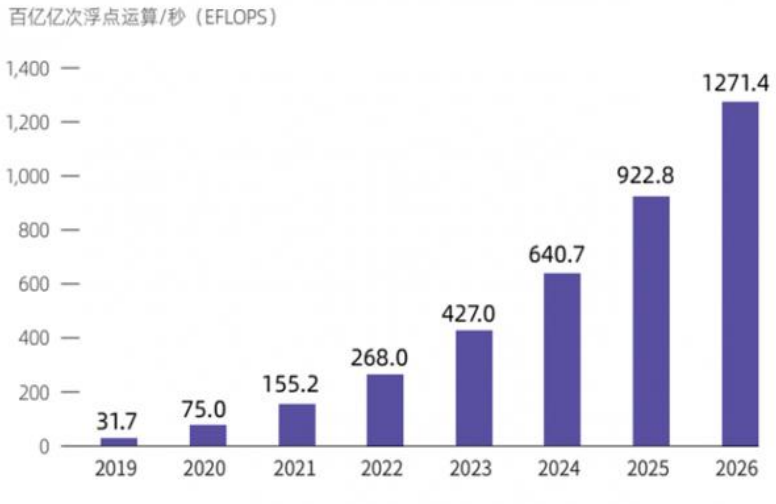


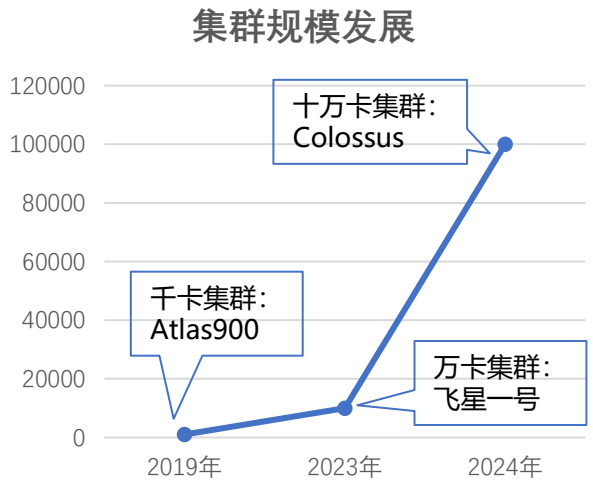
NPU卡

服务器

AI生成

服务器集群



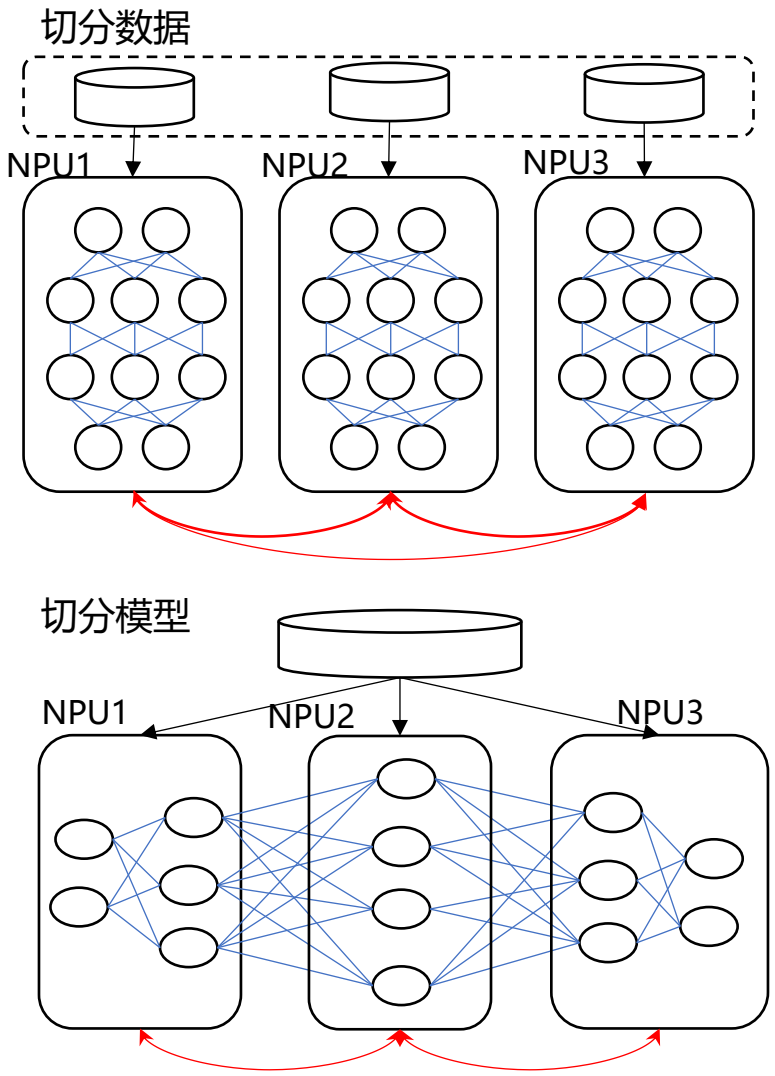


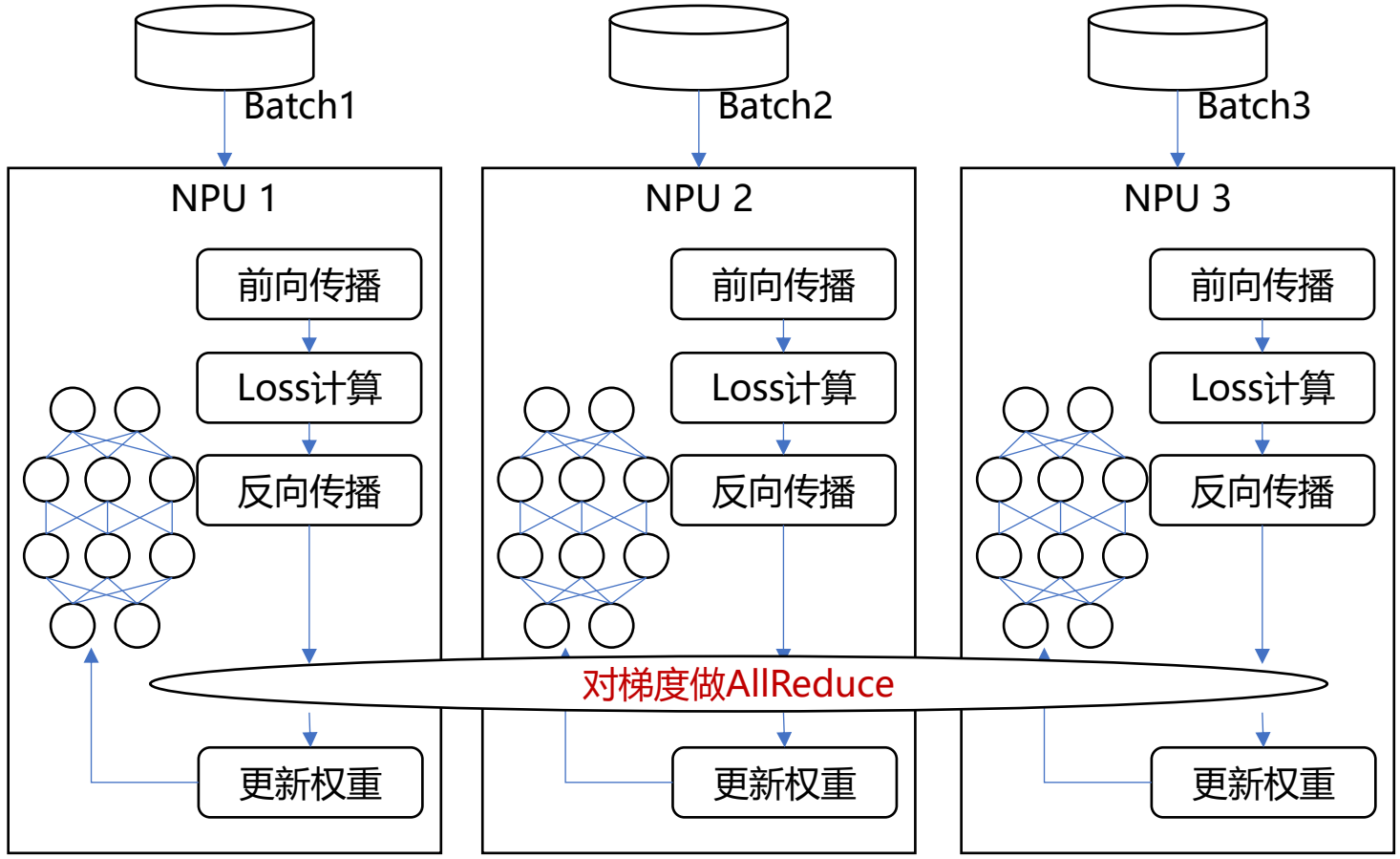
AI进入生成时代，进入爆发增长期。算力发展趋势呈指数级增长，对应的集群规模也呈指数级增长：

- 2019年，Atlas900 千卡服务器集群
- 2023年，科大讯飞发布万卡集群：飞星一号
- 2024年11月，马斯克的xAI建成十万卡GPU集群：Colossus

CANN

分布式AI与集合通信：集群计算需要集合通信



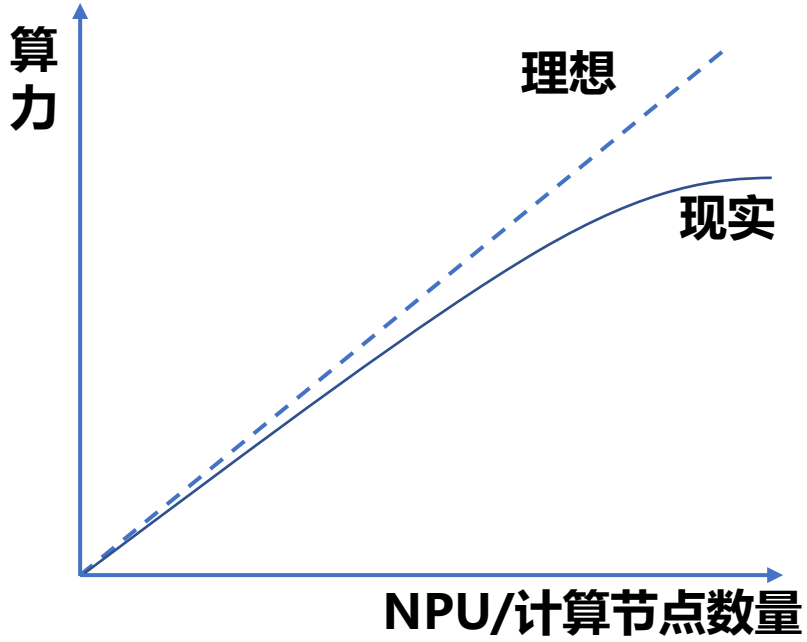


HCCL集合通信是一组通信成员都参与的全局通信操作

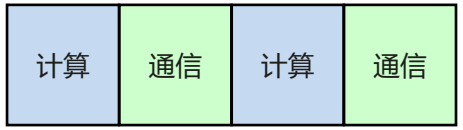
CANN

集群计算的挑战：集群规模增加，集合通信性能直接影响计算效率

挑战：集群规模越大，有效算力并不线性增长。

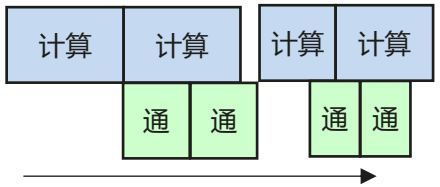


常规模型训练/推理方案



$$效率 = \frac{计算时间}{计算时间 + 通信时间}$$

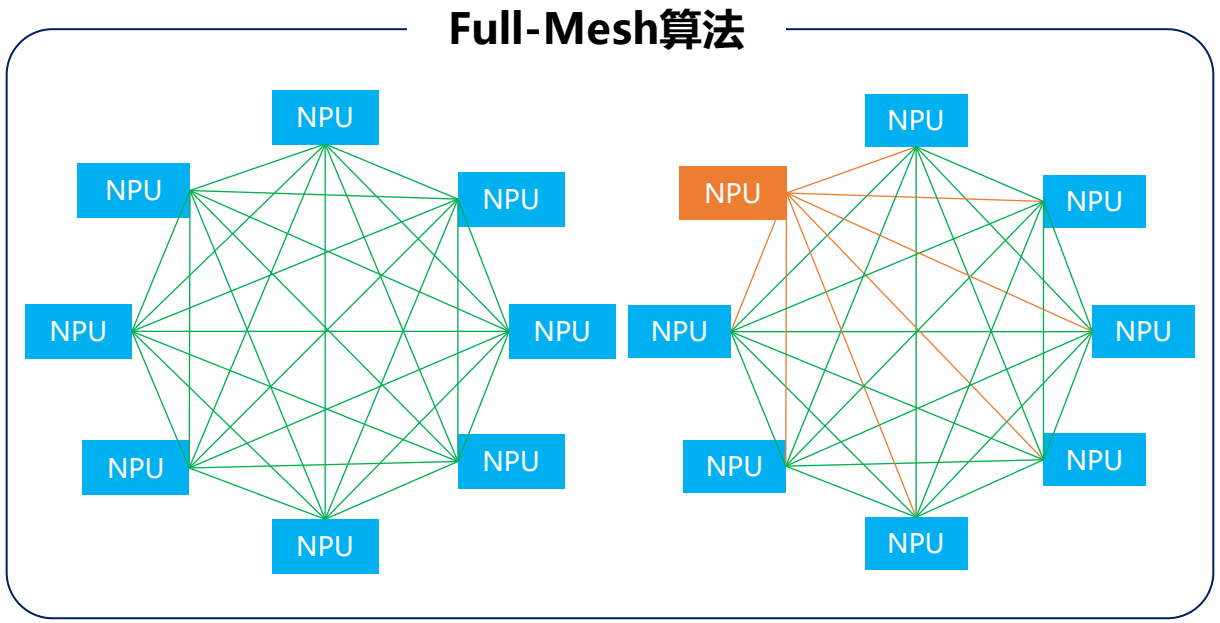
优化思路1：计算与通信流水并行

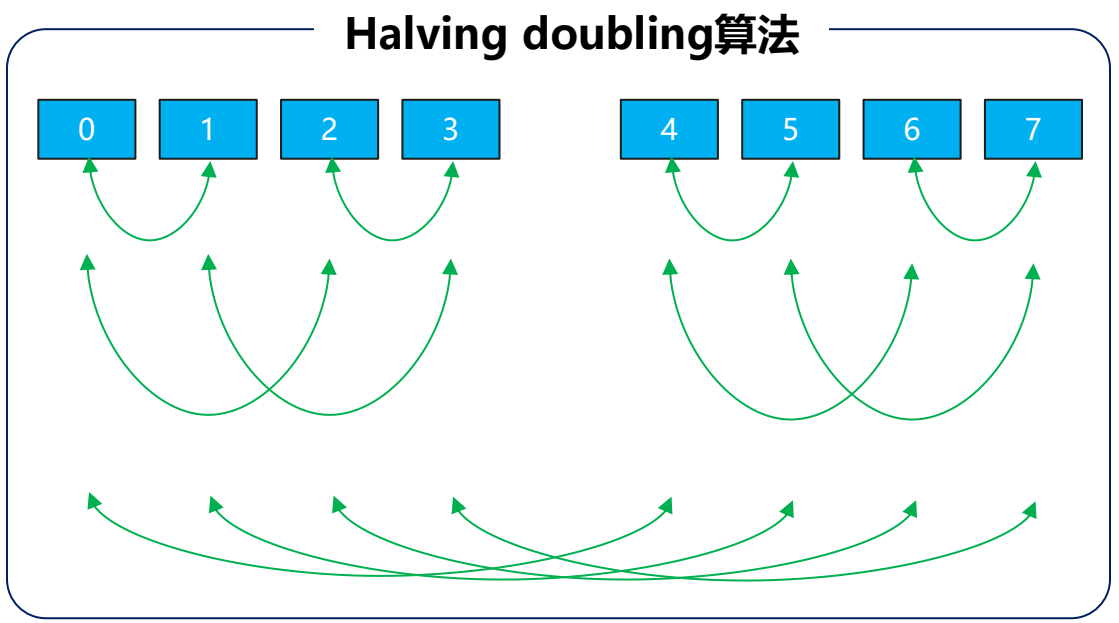


优化思路2：缩短通信时间

CANN

集合通信性能：一个通信算子有不同通信算法，不同算法适用于不同场景/拓扑





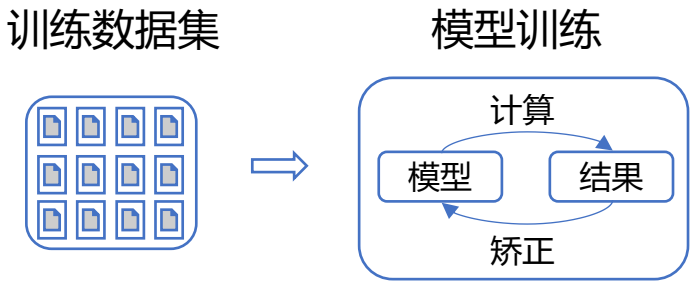
- 适用拓扑：Full-mesh（**小范围低时延通信**）
- 1步通信，可同时访问其它所有NPU

- 适用拓扑：Fat-Tree（**大规模对称通信**）
- log_2N步通信，对大集群友好

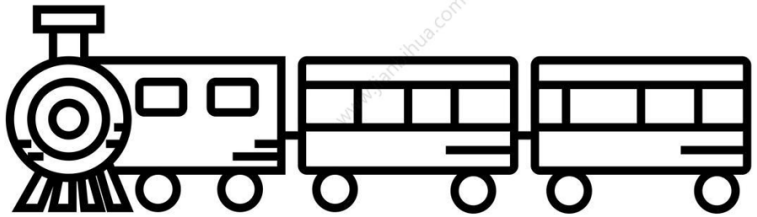
CANN

HCCL优势：针对不同通信业务诉求，选择差异化通信引擎

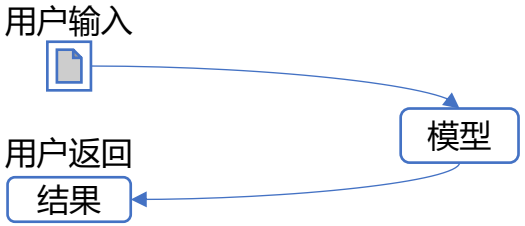
训练：完成海量数据集计算，通信关注**高吞吐**



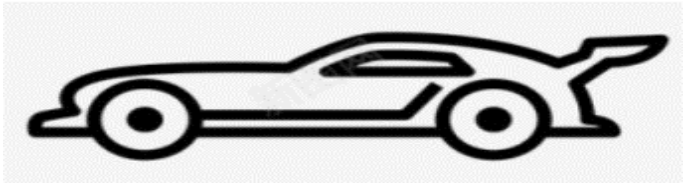
TS调度DMA通信：大数据量重载通信，带宽利用率高



推理：尽快为客户返回结果，通信关注**低时延**



AIV计算核通信：小数据量低时延通信

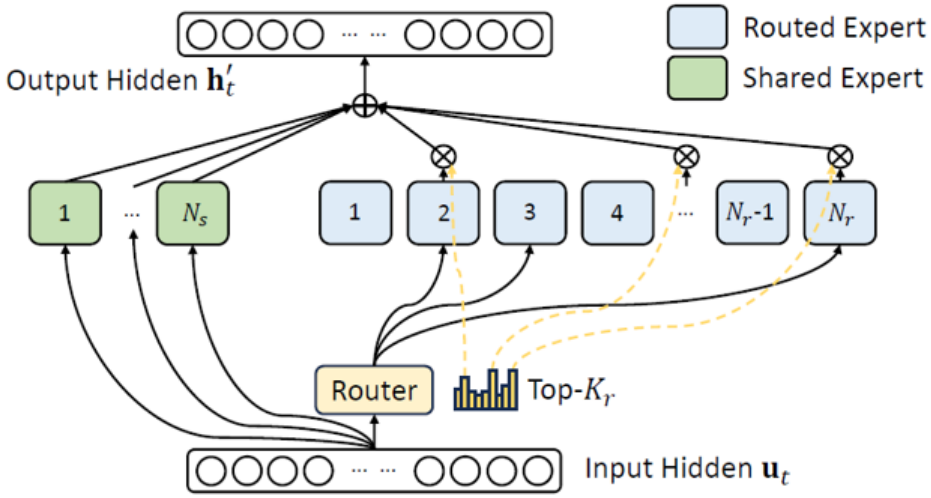


HCCL可根据通信数据量大小等因素，自动选择最优通信引擎

CANN

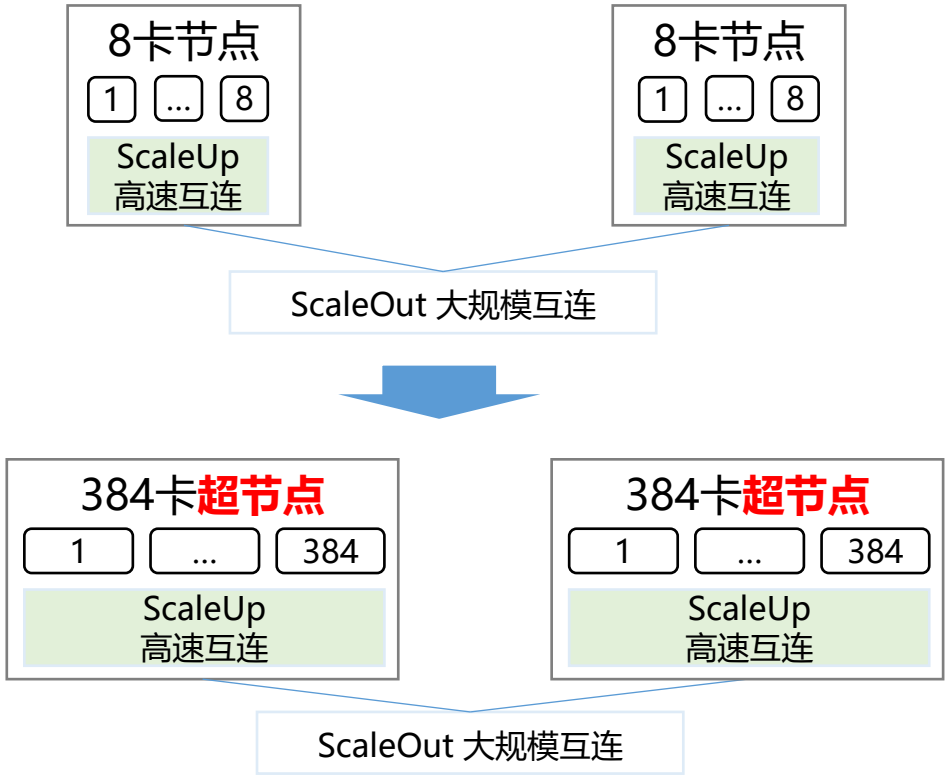
HCCL优势：结合昇腾超节点硬件，充分发挥硬件通信性能

模型规模膨胀，需切分部署到更多芯片上



以DeepSeek的MoE专家并行为例，需部署288个专家
对通信提出了更高的性能要求

超节点大幅提升高速互连规模，以应对大模型挑战

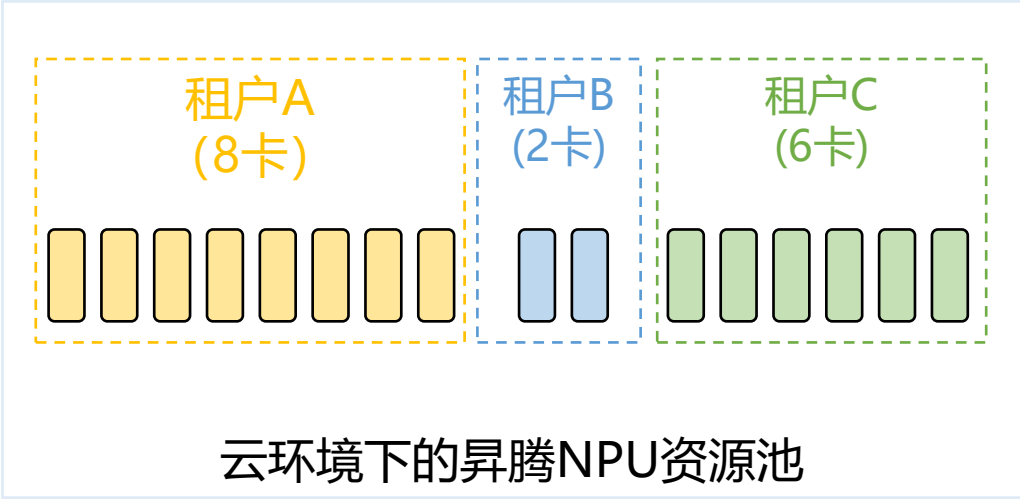


HCCL充分发挥超节点通信优势，支持流水并行、专家并行等主流大模型切分与并行方案

CANN

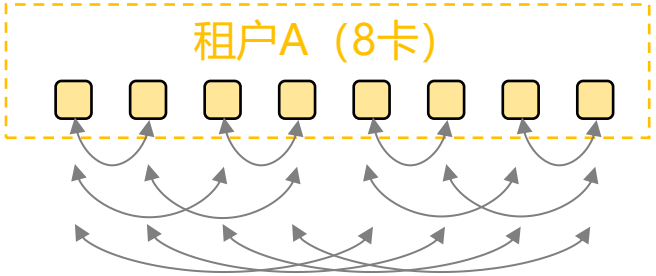
HCCL优势：支持非规则拓扑的NHR通信算法

多租户场景下出现非规则拓扑

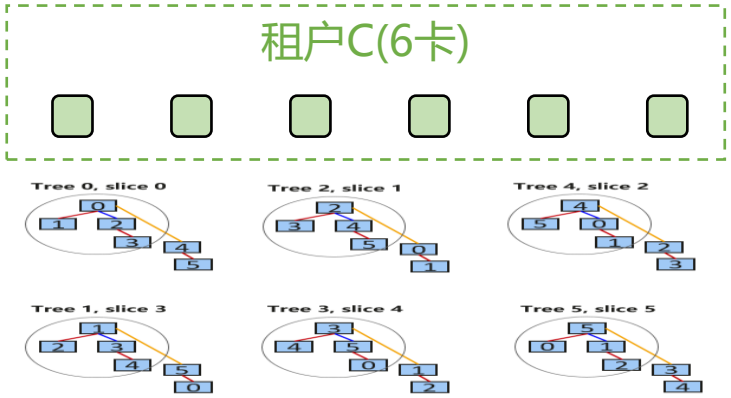


不同拓扑环境下使用不同通信算法

2的N次幂规则拓扑，可选用通用HD算法



非2的N次幂不规则拓扑，自动选择专用NHR算法

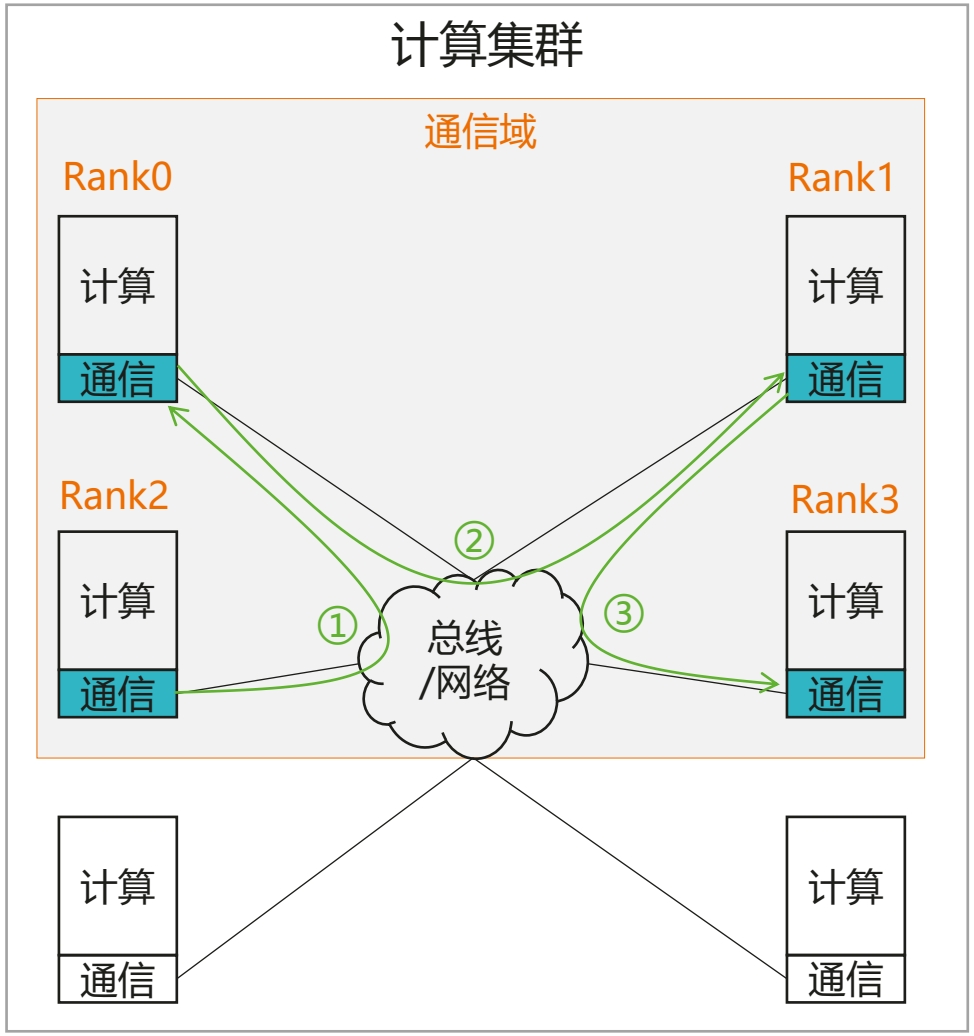


HCCL根据网络拓扑选择最优通信算法，充分发掘硬件潜力

CANN

目录

HCCL使用：基础概念-通信域组织通信成员，通信算子承载通信算法



通信成员：参与通信的**最小逻辑实体**，可以是一颗芯片、一个Die、甚至一个进程，一般称为Rank。

通信域：一组通信成员的组合，描述**通信范围**。一个计算集群可创建多个通信域，一个通信成员可加入多个通信域。

通信算子：**在通信域内完成通信任务的算子**，集合通信指所有成员一起参与的通信操作，如Broadcast。通信域内通信算子串行执行。

通信算法：完成**通信算子的实现方案**，不同拓扑、数据量下可能有不同的通信算法方案，实现不同**流量规划策略**。一个通信算法中一般包括多个通信步骤。

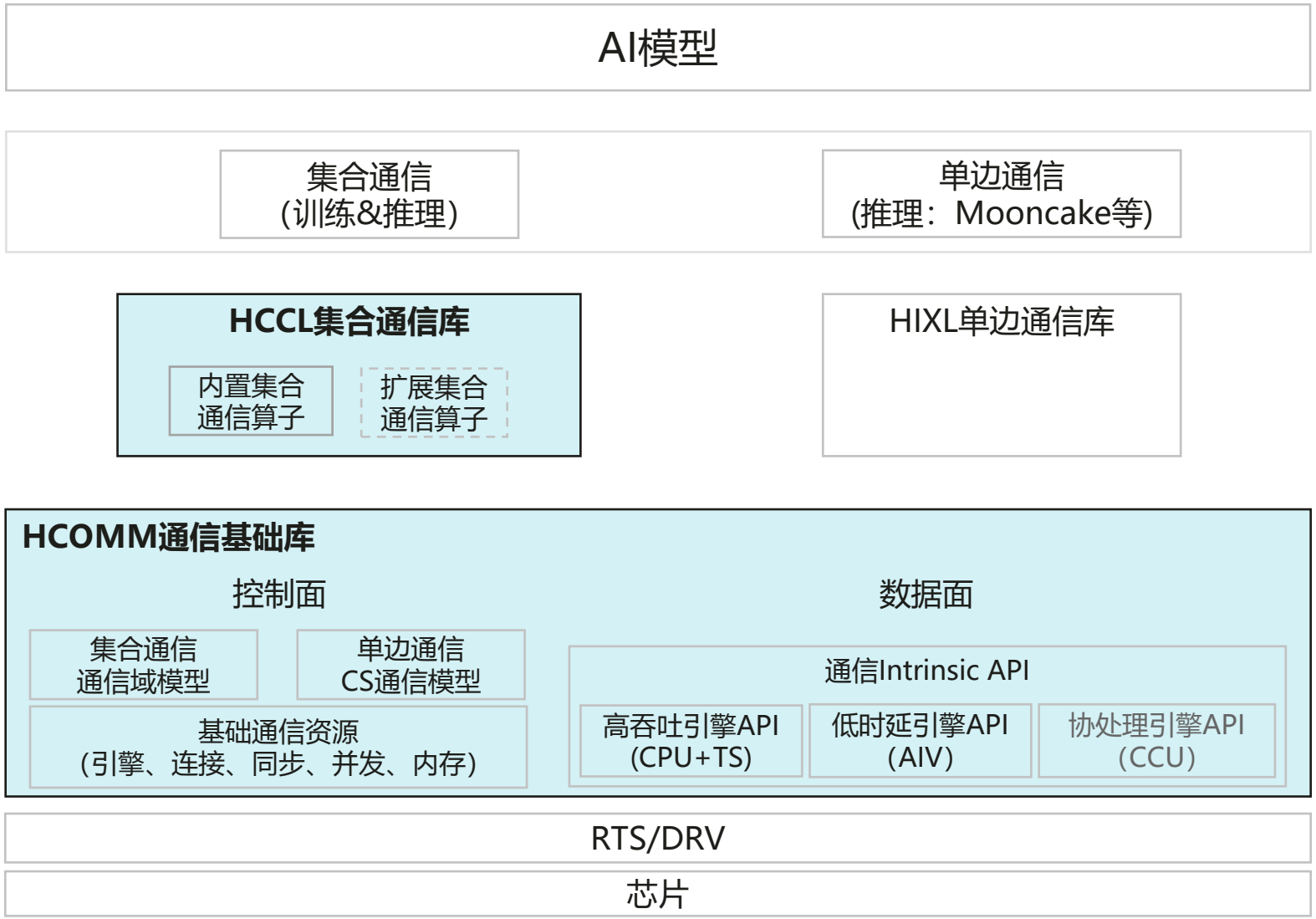
通信域

通信算子

Ring通信算法

CANN

CANN通信库架构：基础库提供芯片基础通信能力，通信库提供差异化通信服务



CANN通信库为不同业务场景提供差异化通信服务，对接主流计算框架，支持客户模型平滑迁移。

HCCL集合通信库：

1、**分层解耦**：通信算子与通信域管理分离，支持自定义通信算子灵活扩展。

2、**全面开源**：开放芯片底层基础通信能力，支持第三方灵活创新。

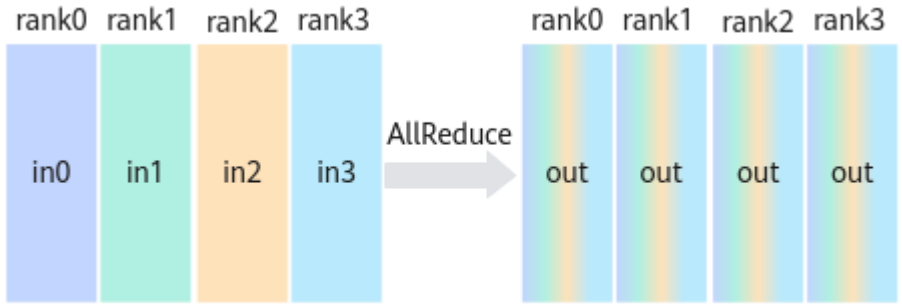
https://gitcode.com/cann

CANN

HCCL使用：集合通信算子举例

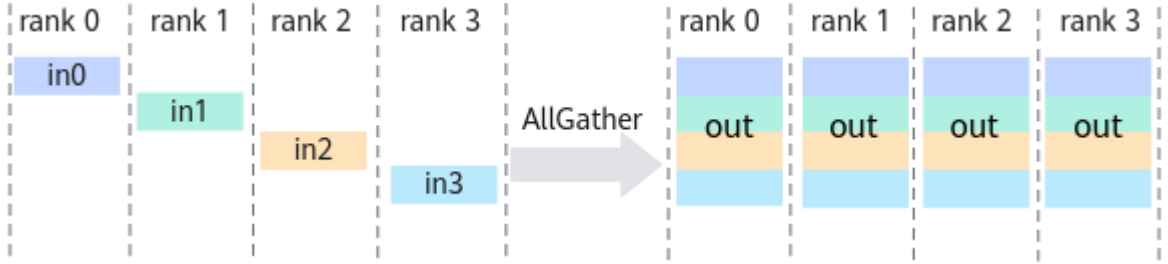
AllReduce：

收集通信域内所有节点数据并做Reduce操作



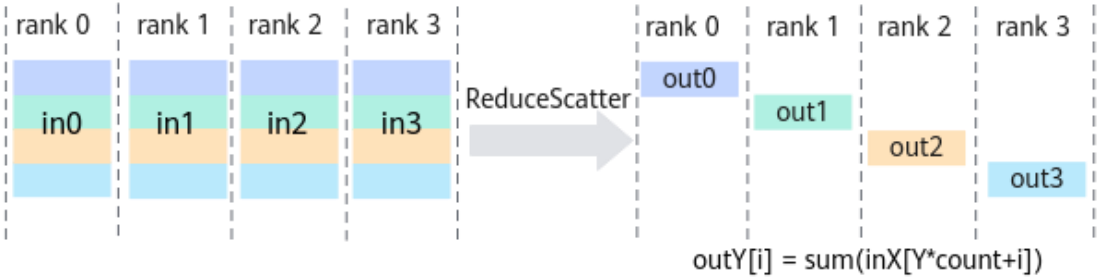
AllGather：

每个节点收集通信域内所有节点的数据



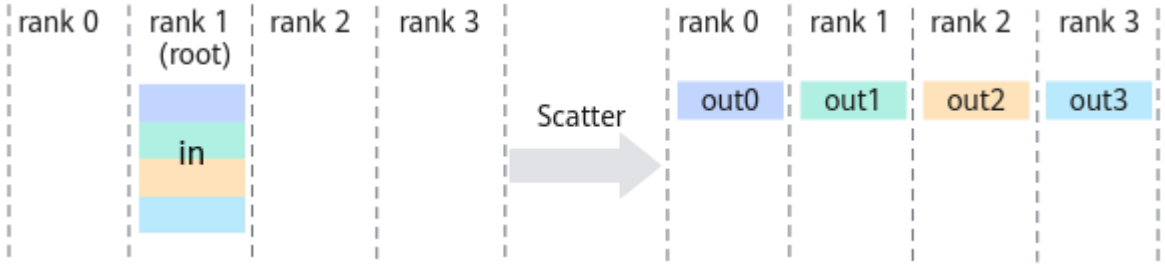
ReduceScatter：

通信域内所有节点数据做Reduce操作后均分至所有节点



Scatter：

将通信域内root节点的数据均分至所有节点



CANN

HCCL使用：集合通信算子接口简介

- **函数原型**

  HcclResult HcclAllReduce(void *sendBuf, void *recvBuf, uint64_t count, HcclDataType dataType, HcclReduceOp op,
  HcclComm comm, aclrtStream stream)

- **功能说明**

  集合通信算子AllReduce的操作接口，将group内所有节点的同名张量进行reduce操作后，再把结果发送到所有节点的输出buffer，其中reduce操作类型由dataType参数指定。

- **参数说明**

参数名	输入/输出	描述
sendBuf	输入	源数据buffer地址。
recvBuf	输出	目的数据buffer地址，集合通信结果输出至此buffer中。
count	输入	参与allreduce操作的数据个数，比如只有一个int32数据参与，则count=1。
dataType	输入	allreduce操作的数据类型。
op	输入	reduce的操作类型，目前支持操作类型为sum、prod、max、min。
comm	输入	集合通信操作所在的通信域。
stream	输入	本rank所使用的stream。

HCCL快速上手：安装

1. 安装昇腾NPU驱动包

./Ascend-hdk-<chip_type>-npu-driver.run –full

2. 安装昇腾NPU固件包

./Ascend-hdk-<chip_type>-npu-firmware.run --full

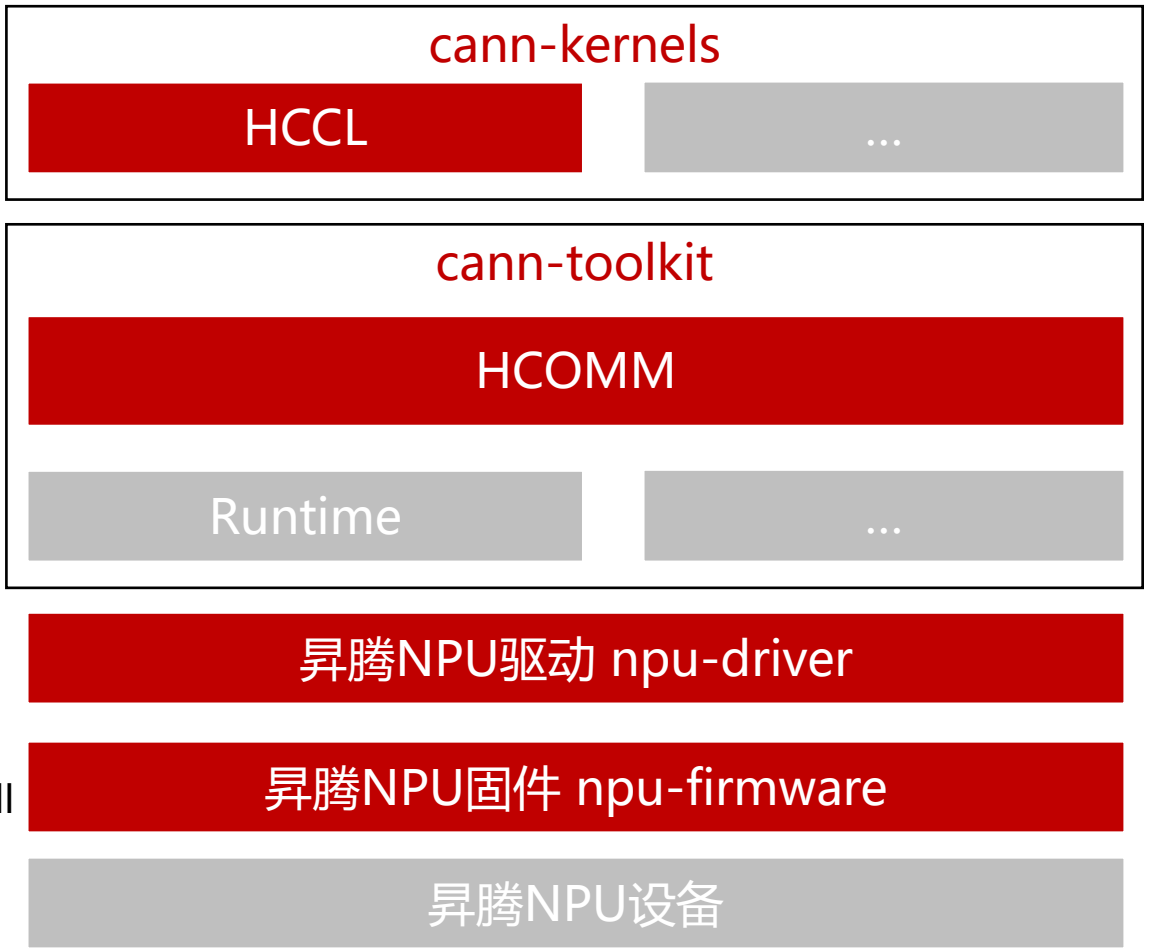
3. 安装CANN Toolkit开发套件包

./Ascend-cann-toolkit_<version>_linux.run –install
source /usr/Ascend/ascend-toolkit/set_env.sh

4. 安装算子包（以Atlas A3系列产品为例）

./Atlas-A3-cann-kernels_<version>_linux-<arch>.run --install

软件安装指南，详见：
https://hiascend.com/document/redirect/CannCommunityInstWizard



HCCL快速上手：使用hccltest工具测试

1. 编译安装MPI

./configure --disable-fortran --prefix=/usr/local/mpich --with-device=ch3:nemesis

2. 编译构建hccl_test工具

make MPI_HOME=/usr/local/mpich ASCEND_DIR=${ASCEND_HOME_PATH}

3. 调用HCCL集合通信算子

mpirun -n 8 ./bin/all_reduce_test -b 64m -e 512m -d fp32 -o sum -

```
[root@bj29 hccl_test]# mpirun -n 8 ./bin/all_reduce_test -b 64m -e 512m -f 2 -d fp32 -o sum -p 8
```

HcclTest工具使用介绍，详见：
https://hiascend.com/document/redirect/CannCommunityToolHcclTest

CANN

HCCL快速上手：通信算子调用

```
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();
   }
   ...
}
```

以1机4卡为例，启动4个线程，每个线程分别执行以下流程调用AllReduce算子

0. 设置当前线程操作的设备
   aclrtSetDevice(device_id);

1. 初始化通信域
   HcclCommInitRootInfo(rankSize, &rootInfo, rank, &comm);

2. 申请资源（Stream、Memory等）
   aclrtCreateStream(&stream);
   aclrtMalloc(&sendBuf, mallocSize, ACL_MEM_MALLOC_HUGE_FIRST);
   aclrtMalloc(&recvBuf, mallocSize, ACL_MEM_MALLOC_HUGE_FIRST);

3. 调用HCCL集合通信算子
   HcclAllReduce(sendBuf, recvBuf, count, HCCL_DATA_TYPE_FP32,
   HCCL_REDUCE_SUM, comm, stream);

4. 同步等待执行完成
   aclrtSynchronizeStream(stream);

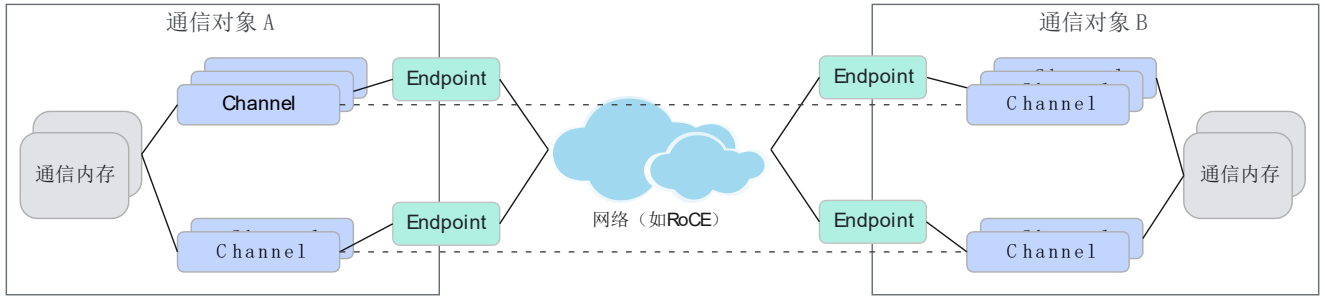
参见代码example：
https://gitcode.com/cann/hccl/tree/master/examples/02_collectives/01_allreduce

CANN

目录

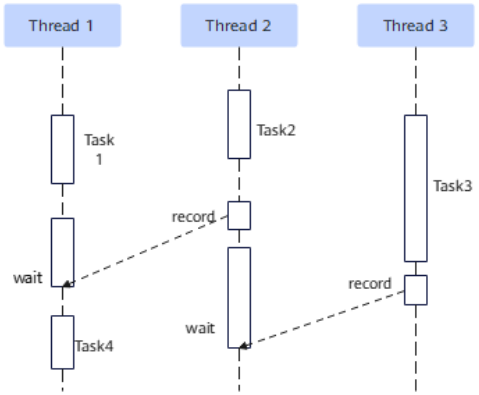
HCCL算子开发：通信算子编程模型

通信模型



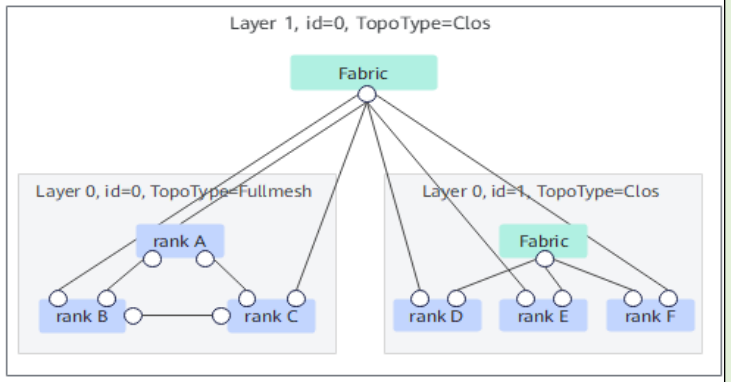
Channel：
- 本端通信对象基于特定Endpoint与远端通信对象特定Endpoint之间建立的通信通道；
- 用户使用Channel可以读写远端通信对象内存或与远端通信对象进行同步。

并发模型



- 并发单元抽象为Thread，通信任务与Thread绑定，不同Thread上的任务可并发执行
- Thread包含多个Notify，Thread间可以通过Notify进行同步；

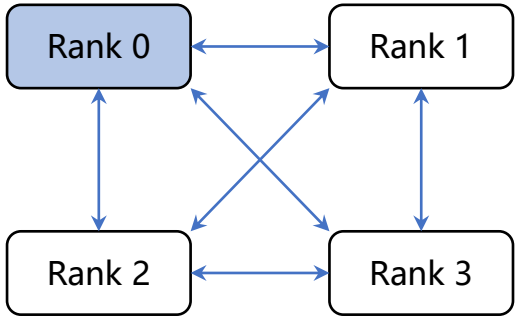
拓扑模型



以Graph的方式对分级拓扑进行建模，并提供查询接口

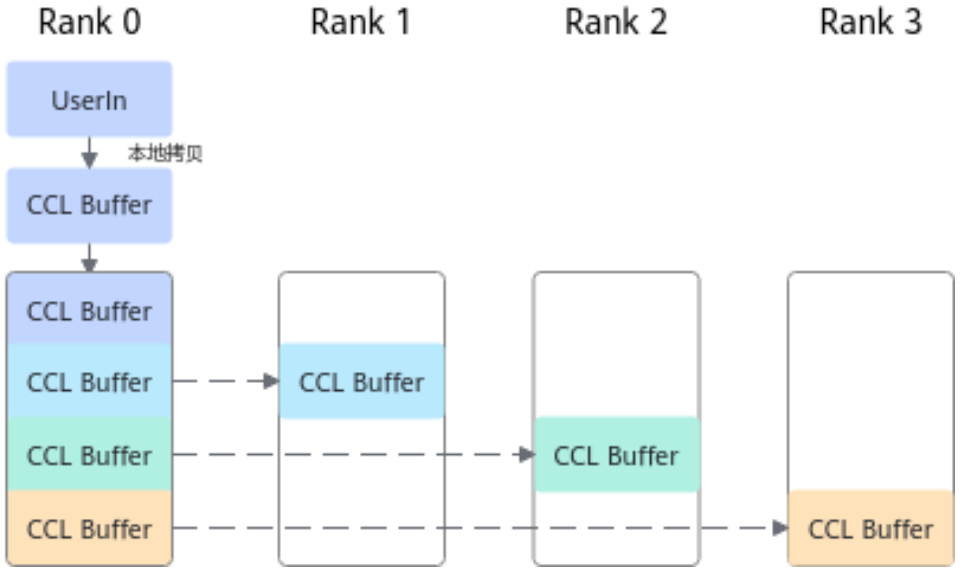
类型		通信编程API样例	说明
控制面	拓扑管理	HcommGetNetLayers (comm, **layers, *num); HcommGetRanksByNetLayer (comm，layer, *rankList, *rankNum);	获取topo信息
	并发资源	HcommAllocThread(comm, engine, threadNum, notifyNumPerThread, *thread)	获取Thread
	内存注册	HcomRegMem(comm, tag, *mem, *handle)	内存注册
	链接资源	HcommChannelCreate(comm, tag, engine, *channelDesc, channelNum, *channel)	创建通信channel
数据面	同步	HcommLocalRecord(thread, dstThread, notify);	thread间发送同步信号
		HcommLocalWait(thread, notify, timeout);	等待其他thread同步信号
		HcommNotifyRecord(thread, channel, notify)	Rank间发送远程同步信号
		HcommNotifyWait(thread, channel, notify, timeout);	等待Rank间同步信号
	传输	HcommLocalCopy(thread, dst, src, len)	本地数据拷贝
		HcommWrite(thread, channel, dst, src, len);	向远端写数据
		HcommRead(thread, channel, dst, src, len);	从远端读数据

HCCL算子开发：以Scatter的Mesh算法为例



Scatter算子：Root节点将自身数据平均分发给所有Rank

Mesh算法：Root节点可以与其他Rank直接通信



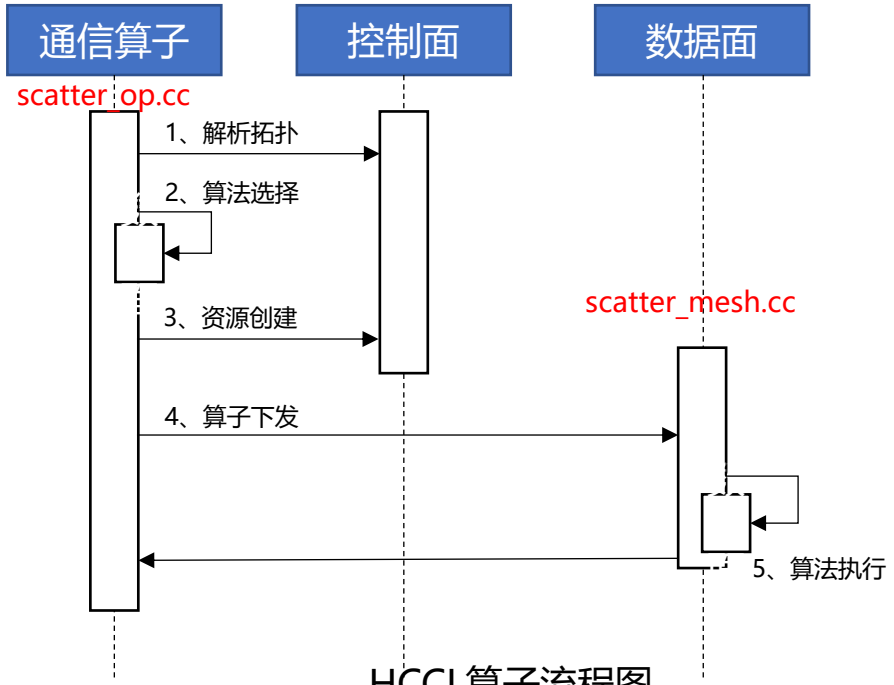
以Rank 0（Root）为视角的Mesh算法步骤示意：

1、用户内存拷贝到本地通信内存（CCL Buffer）

2、从本地通信内存拷贝到其他对端的通信内存

3、各卡从通信内存拷贝到输出内存

HCCL算子流程：控制面准备通信资源，数据面执行数据搬运流程



HCCL算子流程图

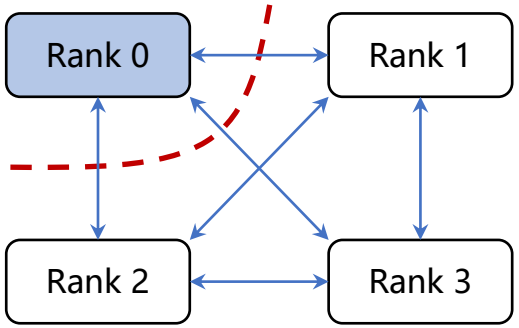
```
src
└── ops
    ├── aicpu                          aicpu launch接口实现目录
    ├── channel                        建链计算接口实现目录
    ├── topo                           topo解析实现目录
    └── scatter                        scatter算子实现目录
        ├── scatter_op.cc              scatter算子调用入口和控制面实现
        └── algo                       scatter算子数据面实现
            ├── template
            │   ├── scatter_mesh.cc    子通信域具体算法实现
            │   └── ...
            ├── scatter_mesh_executor.cc   算法组合实现
            └── ...
```

HCCL算子相关代码结构

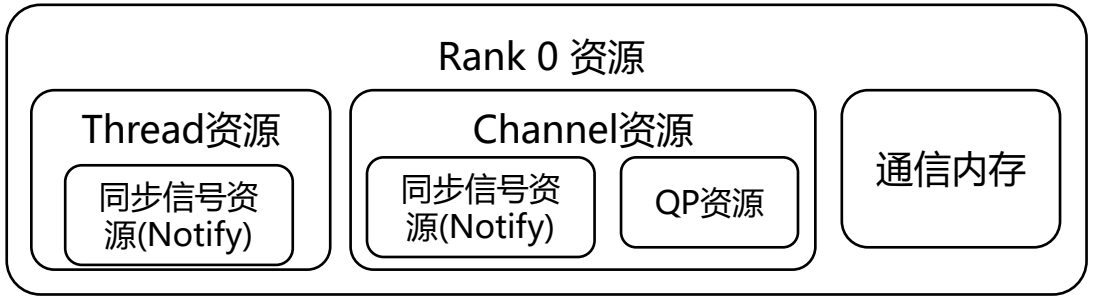
以Scatter算子为例，涉及修改的文件：

- **控制面**实现（scatter_op.cc）
  - 拓扑查询：查询当前网络拓扑信息
  - 算法选择（可选）：根据拓扑信息，选择合适的算法。例如，Fullmesh拓扑选择Mesh算法；CLOS的物理拓扑选择NHR算法/Ring算法等；
  - 资源创建：根据所用的算法，申请需要的通信资源（资源包括Channel、Thread、同步信号资源等
- **数据面**实现（scatter_mesh.cc）
  - 算子下发：将算子下发到执行引擎上
  - 算法执行：通信引擎执行使用数据面API实现的算法流程

HCCL算子开发：算子控制面申请通信算法使用的资源



- 以Rank0为视角：与3个对端通信，需建3个Channel
- 自身操作以及与其他Rank交互可并发，需要1个主Thread和4个从Thread；
- 主Thread需要4个Notify，与从Thread同步；
- 从Thread需要1个Notify与主Thread同步。



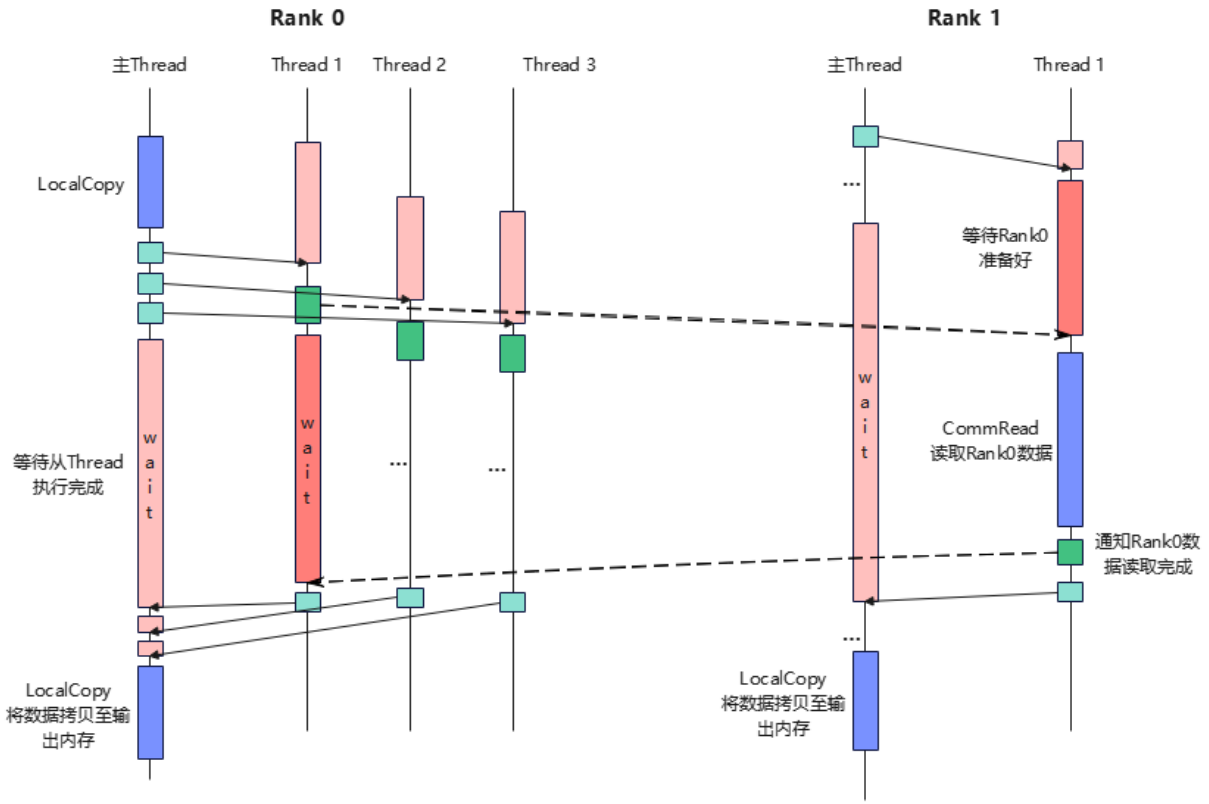
Scatter_op.cc代码示意

```
HcclResult AllocAlgResource(HcclComm comm, const OpParam& param, AlgResourceRequest &resRequest)
{
  CommBuffer cclBuffer;
  // 从通信域获取CCL buffer，创建通信域时自动创建CCL buffer
  CHK_RET(CommGetHcclBuffer(comm, &cclBuffer));

  // 创建5个thread，主thead上有4个notify
  // 从thread有4条，每个从thread上有1个notify
  CHK_RET(CommAllocThreadRes(comm, param.engine, 1, 4, masterThread));
  CHK_RET(CommAllocThreadRes(comm, param.engine, 4, 1, threads));

  // 获取建链诉求
  std::vector<ChannelDesc> &levelNChannelRequest = resRequest.channels;
  // 获取建链数量
  u32 validChannelNum = levelNChannelRequest.size();

  // 创建链路，结果放在levelNChannels数组中
  std::vector<ChannelHandle> levelNChannels;
  levelNChannels.resize(validChannelNum);
  CHK_RET(CommChannelCreate(comm, param.algTag, param.engine, levelNChannelRequest.data(),
          validChannelNum, levelNChannels.data()));

  return HCCL_SUCCESS;
}
```


CANN

HCCL算子开发：算子数据面实现数据搬运步骤

Rank 0（Root）与Rank的数据面交互流程



Scatter_Mesh.cc代码示意

```
HcclResult ScatterExecutorBase::Orchestrate(const OpParam &param, AlgResourceCtx* resCtx)
{
  // 省略解析算子参数和资源信息
  // root rank拷贝数据到CCLBuffer
  if (userRank == root) {
    for (u32 i = 0; i < userRankSize; i++) {
      void* src = (u8*)inputPtr + recvSize * i;
      void* dst = (u8*)CCLBufferPtr + recvSize * i;
      CHK_RET(CommLocalCopy(masterThread, dst, src, recvSize));
    }
  }
  // 主流通知从流
  for (u32 i = 0; i < userRankSize; i++) {
    CommLocalNotifyRecord(masterThread, threads[i], 0);
    CommLocalNotifyWait(threads[i], 0, CUSTOM_TIMEOUT)
  }
  // 计算偏移地址
  u64 offset = recvSize * userRank;
  // root rank向其他rank发送同步信号
  if (userRank == root) {
    for (u32 dstRank = 0; dstRank < rankSize; dstRank++) {
      CHK_RET(CommNotifyRecord(threads[dstRank], channels[dstRank].handle, 0));
      CHK_RET(CommNotifyWait(threads[dstRank], channels[dstRank].handle, 1, CUSTOM_TIMEOUT));
    }
  } else { // 非root rank 从root rank读取数据
    CHK_RET(CommNotifyWait(threads[userRank], channels[srcRank].handle, 0, CUSTOM_TIMEOUT));
    s8 * src = static_cast<s8 *>(channels[srcRank].remoteAddr);
    CHK_RET(CommRead(threads[userRank], channels[srcRank].handle, (u8*)CCLBufferPtr + offset,
      src + offset, recvSize));
    CHK_RET(CommNotifyRecord(threads[userRank], channels[srcRank].handle, 1));
  }
  // 从流通知主流
  for (u32 i = 0; i < userRankSize; i++) {
    CommLocalNotifyRecord(threads[i], masterThread, i);
    CommLocalNotifyWait(masterThread, i, CUSTOM_TIMEOUT)
  }
  // 从CCL拷贝数据到output
  CHK_RET(CommLocalCopy(masterThread, outputPtr, (u8*)CCLBufferPtr + offset, recvSize));
}
```

HCCL开源社区已上线，欢迎大家参与共创

开源代码仓

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

hcomm（CANN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