CANN
6.3.RC2

集合通信 OpBase 接口参考

文档版本 01

发布日期 2023-08-04

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 简介

HCCL（Huawei Collective Communication Library）是一个集合通信库，可以集成到分布式机器学习框架中，实现针对华为昇腾AI处理器单机多卡/多机多卡的高性能集合通信功能。提供all-reduce、all-gather、broadcast、reduce-scatter等通信功能，实现昇腾AI处理器间的高速互联。

您可以在“CANN软件安装目录/include/hccl”下查看接口定义。框架开发者可以通过这套C接口，实现OPBase模式下的框架适配，实现分布式能力。

2 HcclCommInitClusterInfo

函数原型

HcclResult HcclCommInitClusterInfo(const char *rankTable, uint32_t rank, HcclComm *comm);

功能说明

基于ranktable初始化HCCL通信域。

参数说明

参数名	输入/输出	描述
rankTable	输入	rank table的文件路径（含文件名），作为字符串最大长度为4096字节，含结束符。
rank	输入	本rank的rank id。
comm	输出	将初始化后的通信域以指针的信息回传给调用者。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

- 重复初始化会报错

3 HcclGetRootInfo

函数原型

HcclResult HcclGetRootInfo(HcclRootInfo *rootInfo);

功能说明

在初始化HCCL（HcclCommInitRootInfo）前应调用HcclGetRootInfo，生成此rank的标识信息（HcclRootInfo）。然后以此rank为root节点，将此rank的标识信息广播至集群中的所有rank，使用接收到的rootInfo进行HCCL初始化（HcclCommInitRootInfo）。

参数说明

参数名	输入/输出	描述
rootInfo	输出	本rank的标识信息，主要包含device ip，device id等信息，此信息需广播至集群内所有rank用来进行HCCL初始化。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

1. 多机集合通信场景，调用HcclGetRootInfo前，需要配置环境变量HCCL_IF_IP、HCCL_WHITELIST_DISABLE、HCCL_WHITELIST_FILE。
2. 该接口和HcclCommInitRootInfo配对使用，不能单独使用。

4 HcclCommInitRootInfo

函数原型

HcclResult HcclCommInitRootInfo(uint32_t nRanks, const HcclRootInfo *rootInfo, uint32_t rank, HcclComm *comm);

功能说明

根据rootInfo初始化HCCL，创建HCCL通信域。

参数说明

参数名	输入/输出	描述
nRanks	输入	集群中的rank数量。
rootInfo	输入	root rank信息，主要包含root rank的ip，id等信息，由HcclGetRootInfo()生成。
rank	输入	本rank的rank id。
comm	输出	初始化后的通信域指针。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

1. 所有rank的nranks、rootInfo均应相同。
2. 该接口只能串行调用，不支持并发调用。

5 HcclCommDestroy

函数原型

HcclResult HcclCommDestroy(HcclComm comm);

功能说明

销毁指定的HCCL 通信域。

参数说明

参数名	输入/输出	描述
comm	输入	销毁该通信域。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

无

6 HcclAllReduce

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

集合通信域all-reduce操作接口，将所有rank的sendBuf相加（或其他操作）后，再把结果发送到所有rank的recvBuf。

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。 针对昇腾910 AI处理器，支持数据类型int8/int32/fp16/fp32。 针对昇腾910B AI处理器，支持数据类型int8/int16/int32/fp16/fp32。
op	输入	reduce的操作类型，目前支持sum/prod/max/min等操作类型。
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

所有rank的count、dataType、op均应相同。

7 HcclBroadcast

函数原型

HcclResult HcclBroadcast(void *buf, uint64_t count, HcclDataType dataType, uint32_t root, HcclComm comm, aclrtStream stream);

功能说明

集合通信域Broadcast操作接口，将root节点的数据广播到其他rank。

参数说明

参数名	输入/输出	描述
buf	输入/输出	数据buffer，对于root节点，是数据源；对于非root节点，是数据接收buffer。
count	输入	参与broadcast操作的数据个数，比如只有一个int32数据参与，则count=1。
dataType	输入	broadcast操作的数据类型。 针对昇腾910 AI处理器，支持数据类型int8, uint8, uint16, int32, uint32, int64, uint64, float16, float32, float64。 针对昇腾910B AI处理器，支持数据类型int8, uint8, int16, uint16, int32, uint32, int64, uint64, float16, float32, float64。
root	输入	作为broadcast root的rank id。
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

1. 所有rank的count、dataType、root均应相同；
2. 全局只能有1个root节点。

8 HcclAllGather

函数原型

HcclResult HcclAllGather(void *sendBuf, void *recvBuf, uint64_t sendCount, HcclDataType dataType, HcclComm comm, aclrtStream stream);

功能说明

实现all-gather操作接口。将所有rank的sendBuf按rank顺序拼接起来，再把结果发送到所有rank的recvBuf。

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。
sendCount	输入	参与allgather操作的sendBuf的数据size，recvBuf的数据size则等于count * rank size。
dataType	输入	allgather操作的数据类型。 针对昇腾910 AI处理器，支持数据类型int8, uint8,uint16, int32, uint32, int64, uint64, float16, float32, float64。 针对昇腾910B AI处理器，支持数据类型int8, uint8, int16, uint16, int32, uint32, int64, uint64, float16, float32, float64。
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

无

9 HcclReduceScatter

函数原型

HcclResult HcclReduceScatter(void *sendBuf, void *recvBuf, uint64_t recvCount, HcclDataType dataType, HcclReduceOp op, HcclComm comm, aclrtStream stream);

功能说明

集合通信域reducescatter操作接口。将所有rank的sendBuf相加（或其他操作）后，再把结果按照rank编号均匀分散的到各个rank的recvBuf。

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。
recvCount	输入	参与reducescatter操作的recvBuf的数据size，sendBuf的数据size则等于recvCount * rank size。
dataType	输入	reduce-scatter操作的数据类型。 针对昇腾910 AI处理器，支持数据类型int8/int32/fp16/fp32。 针对昇腾910B AI处理器，支持数据类型int8/int16/int32/fp16/fp32。
op	输入	reduce的操作类型，目前支持sum/prod/max/min等操作类型。
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

所有rank的recvCount、dataType、op均应相同。

10 HcclReduce

函数原型

HcclResult HcclReduce(void *sendBuf, void *recvBuf, uint64_t count, HcclDataType dataType, HcclReduceOp op, uint32_t root, HcclComm comm, aclrtStream stream);

功能说明

集合通信域reduce操作接口，将所有rank的sendBuf相加（或其他操作）后，再把结果发送到root节点的recvBuf。

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reduce操作的数据个数，比如只有一个int32数据参与，则count=1。
dataType	输入	reduce操作的数据类型。 针对昇腾910 AI处理器，支持数据类型int8/int32/fp16/fp32。 针对昇腾910B AI处理器，支持数据类型int8/int16/int32/fp16/fp32。
op	输入	reduce的操作类型，目前支持sum/prod/max/min等操作类型。
root	输入	作为reduce root的rankid
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

所有rank的count、dataType、op均应相同。

11 HcclBarrier

函数原型

HcclResult HcclBarrier(HcclComm comm, aclrtStream stream);

功能说明

将指定通信域内所有rank的stream阻塞，直到所有rank都下发执行该操作为止。

参数说明

参数名	输入/输出	描述
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

12 HcclGetRankSize

函数原型

HcclResult HcclGetRankSize(HcclComm comm, uint32_t *rankSize);

功能说明

查询当前集合通信域的rank总数。

参数说明

参数名	输入/输出	描述
comm	输入	集合通信操作所在的通信域。
rankSize	输出	指定集合通信域的rank总数输出地址指针。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

无

13 HcclGetRankId

函数原型

HcclResult HcclGetRankId(HcclComm comm, uint32_t *rank);

功能说明

获取device在集合通信域comm中对应的rank序号。

参数说明

参数名	输入/输出	描述
comm	输入	集合通信操作所在的通信域。
rank	输出	指定集合通信域中rank序号的输出地址指针。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

无

14 HcclSend

函数原型

HcclResult HcclSend(void* sendBuf, uint64_t count, HcclDataType dataType, uint32_t destRank,HcclComm comm, aclrtStream stream);

功能说明

集合通信域Send操作接口。将当前rank的sendBuf数据发送至destRank。

参数说明

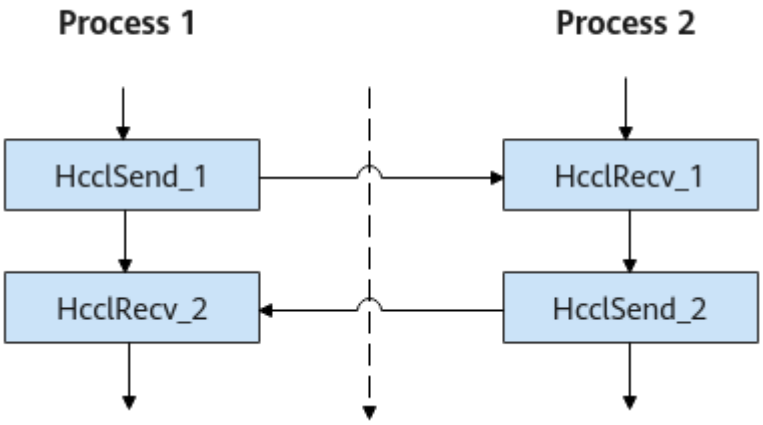
参数名	输入/输出	描述
sendBuf	输入	源数据buffer地址。
count	输入	发送数据的个数。
dataType	输入	发送数据的数据类型。 针对昇腾910 AI处理器，支持数据类型int8, uint8, uint16, int32, uint32, int64, uint64, float16, float32, float64。 针对昇腾910B AI处理器，支持数据类型int8, uint8, int16, uint16, int32, uint32, int64, uint64, float16, float32, float64。
destRank	输入	通信域内数据接收端的rank编号
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

HcclSend与HcclRecv接口采用同步调用方式，且必须配对使用。即一个进程调用HcclSend接口后，需要等到与之配对的HcclRecv接口接收数据后，才可以进行下一个接口调用，如下图所示。



15 HcclRecv

函数原型

HcclResult HcclRecv(void* recvBuf, uint64_t count, HcclDataType dataType, uint32_t srcRank,HcclComm comm, aclrtStream stream);

功能说明

集合通信域Recv操作接口。从srcRank接收数据到当前rank的recvBuf。

参数说明

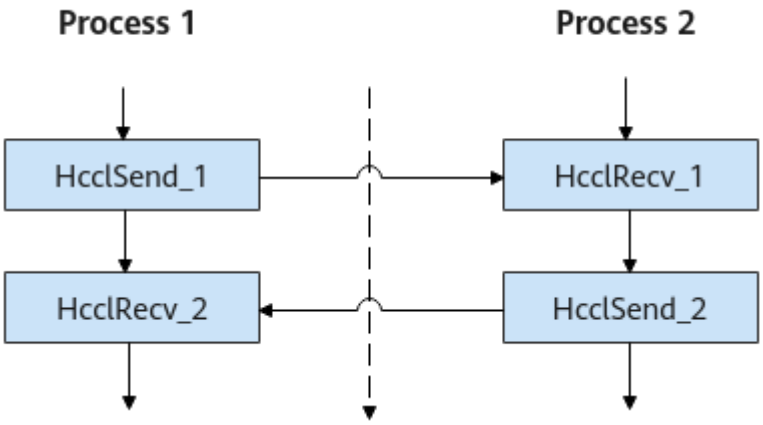
参数名	输入/输出	描述
recvBuf	输入	数据接收buffer地址。
count	输入	接收数据的个数。
dataType	输入	接收数据的数据类型。 针对昇腾910 AI处理器，支持数据类型int8, uint8, uint16, int32, uint32, int64, uint64, float16, float32, float64。 针对昇腾910B AI处理器，支持数据类型int8, uint8, int16, uint16, int32, uint32, int64, uint64, float16, float32, float64。
srcRank	输入	通信域内数据发送端的rank编号
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

HcclSend与HcclRecv接口采用同步调用方式，且必须配对使用。即一个进程调用HcclSend接口后，需要等到与之配对的HcclRecv接口接收数据后，才可以进行下一个接口调用，如下图所示。



16 HcclAlltoAllV

函数原型

HcclResult HcclAlltoAllV(const void *sendBuf, const void *sendCounts, const void *sdispls, HcclDataType sendType, const void *recvBuf, const void *recvCounts, const void *rdispls, HcclDataType recvType,HcclComm comm, aclrtStream stream);

功能说明

集合通信域alltoallv操作接口。向通信域内所有rank发送数据（数据量可以定制），并从所有rank接收数据。

参数说明

参数名	输入/输出	描述
sendBuf	输入	源数据buffer地址。
sendCounts	输入	表示发送数据量的uint64数组，sendCounts[i] = n 表示本rank发给rank i的数据量为n，
sdispls	输入	表示发送偏移量的uint64数组，sdispls[i] = n 表示本rank发给rank i的数据在sendBuf的起始位置相对sendBuf的偏移量，以sendType为基本单位
sendType	输入	若sendType为float32，sendCounts[i] = n表示本rank发给rank i n个float32数据
recvBuf	输出	目的数据buffer地址，集合通信结果输出至此buffer中。
recvCounts	输入	表示接收数据量的uint64数组，recvCounts[i] = n 表示本rank从rank i收到的的数据量为n
rdispls	输入	表示接收偏移量的uint64数组，rdispls[i] = n 表示本rank从rank i的收到数据存放在在recvBuf的起始位置相对recvBuf的偏移量，以recvType为基本单位
recvType	输入	若recvType为float32，recvCounts[i] = n表示本rank从ank i收到n个float32数据

参数名	输入/输出	描述
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。

返回值

HcclResult：接口成功返回HCCL_SUCCESS。其他失败。

约束说明

针对昇腾910 AI处理器，alltoallv的通信域需要满足如下约束：

单server 1p、2p通信域要在同一个cluster内（server内0-3卡和4-7卡各为一个cluster），单server4p、8p和多server通信域中rank要以cluster为基本单位，并且server间cluster选取要一致。

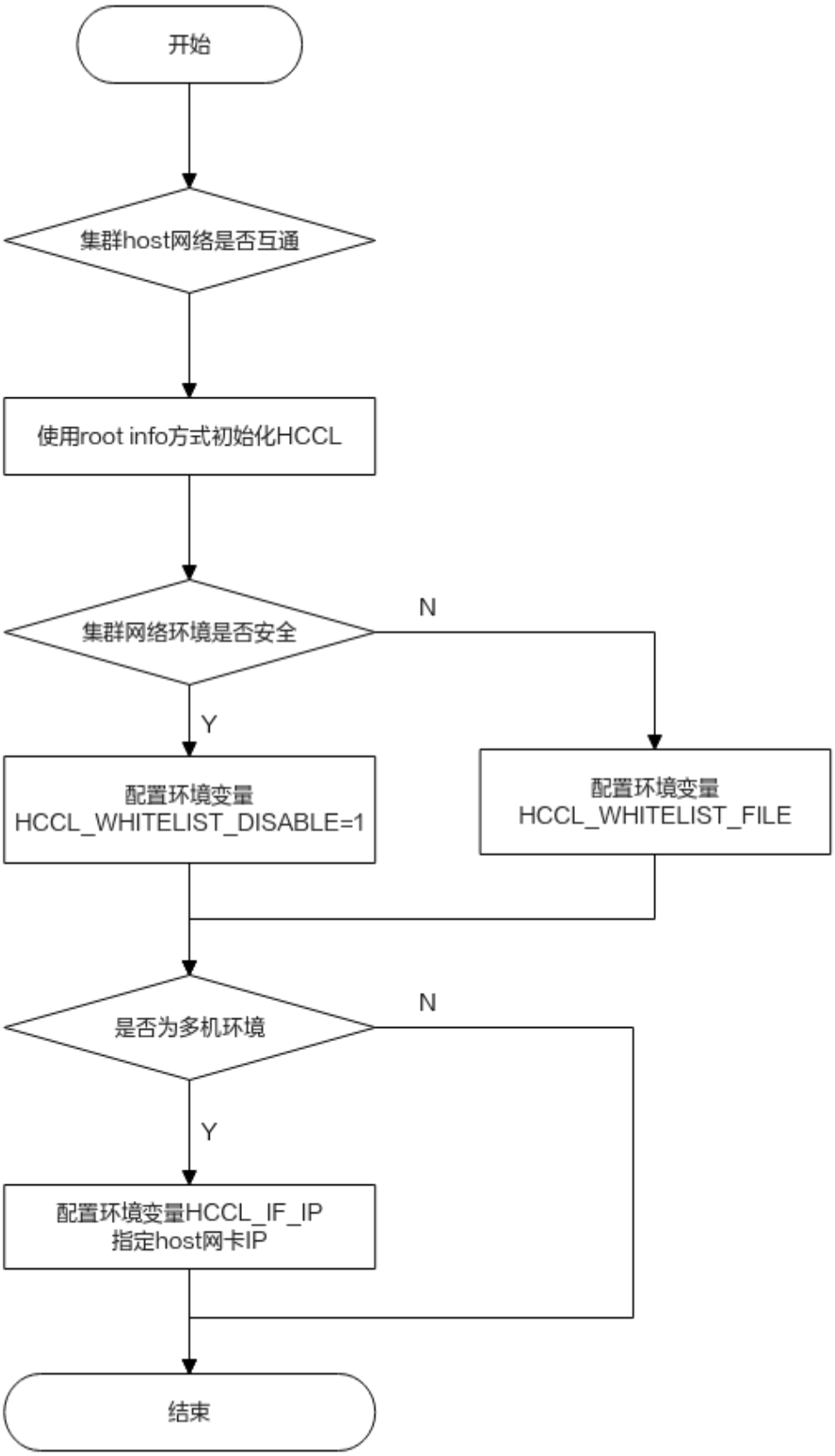
17 HCCL 初始化使用指导

单算子模式下，我们提供两种HCCL初始化流程，分别为：

- 使用ranktable方式初始化：通过ranktable文件指定集群信息，进行集合通信初始化。对应的接口为**2 HcclCommInitClusterInfo**。
- 使用root info方式初始化：指定root rank自动收集集群信息，进行集合通信初始化。对应的接口为**3 HcclGetRootInfo**和**4 HcclCommInitRootInfo**。

对比两种方式，使用root info方式初始化较为简单，在集群host网络互通的情况下，推荐使用。

具体使用流程为：



18 代码样例

```c++
#ifndef __HCCL_TEST_COMMON_H_
#define __HCCL_TEST_COMMON_H_
#include <stdio.h>
#include <string.h>
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
#include "mpi.h"
#include <chrono>
#include "hccl/hccl.h"
#include "acl/acl.h"

#define ACLCHECK(ret) do { \
    if(ret != ACL_SUCCESS)\
    {\
        printf("acl interface return err %s:%d, retcode: %d \n", __FILE__, __LINE__, ret);\
        exit(EXIT_FAILURE);\
    }\
} while(0)

#define HCCLCHECK(ret) do {  \
    if(ret != HCCL_SUCCESS) \
    {   \
        printf("hccl interface return errreturn err %s:%d, retcode: %d \n", __FILE__, __LINE__, ret); \
        exit(EXIT_FAILURE);\
    } \
} while(0)
#endif

bool g_isDevice = false;

bool setup(int device_id) {
  ACLCHECK(aclInit(nullptr));

  ACLCHECK(aclrtSetDevice(device_id));

  aclrtRunMode run_mode;
  ACLCHECK(aclrtGetRunMode(&run_mode));
  bool g_is_device_return = (run_mode == ACL_DEVICE);
  return g_is_device_return;
}

//     请插入调用的通信算子执行函数
// HcclAllReduce 算子sample，通过main函数调用
int hccl_allreduce_sample(HcclComm &hcom) {

    // 指定通信算子调用所需的资源
    int count = 256*256*10;
```



```
    aclrtStream stream;
    ACLCHECK(aclrtCreateStream(&stream));
    aclFloat16 *bufer_host;
    ACLCHECK(aclrtMalloc((void **)&bufer_host, count*sizeof(aclFloat16),
ACL_MEM_MALLOC_HUGE_FIRST));
    aclFloat16 *bufer_dev;
    ACLCHECK(aclrtMalloc((void **)&bufer_dev, count*sizeof(aclFloat16), ACL_MEM_MALLOC_HUGE_FIRST));

    // 获取参与集合通信的rank数量
    unsigned int rankSize = 0;
    HCCLCHECK(HcclGetRankSize(hcom, &rankSize));
    printf("Get rank size is %u \n", rankSize);

    // 获取参与集合通信的rank序号
    unsigned int rank_id = 0;
    HCCLCHECK(HcclGetRankId(hcom, &rank_id));
    printf("Get rank id is %u \n", rank_id);

    // 调用HcclAllReduce接口
    HCCLCHECK(HcclAllReduce(bufer_host, bufer_dev, count, HCCL_DATA_TYPE_FP16, HCCL_REDUCE_SUM,
hcom, stream));
    aclrtSynchronizeStream(stream);

    // 资源析构
    ACLCHECK(aclrtFree(bufer_dev));
    ACLCHECK(aclrtDestroyStream(stream));

    return 0;
}


// sample主函数
int main(int argc, char **argv) {
    int rank, size, device_id;
    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    device_id = rank;
    g_isDevice = setup(device_id);

    HcclComm hcom;
    HcclRootInfo root_info;
    if (rank == 0) {
        // 获取root节点，root节点用户可指定，并非只可以设置为0节点
        HCCLCHECK(HcclGetRootInfo(&root_info));
    }
    MPI_Bcast(&root_info, HCCL_ROOT_INFO_BYTES, MPI_CHAR, 0, MPI_COMM_WORLD);
    // 调用Hccl初始化接口
    HCCLCHECK(HcclCommInitRootInfo(size, &root_info, rank, &hcom));

    int ret = 0;
    // 通信算子执行函数调用，用户可根据需要切换
    ret = hccl_allreduce_sample(hcom);
    if (ret != 0) {
        printf("return error");
        exit(EXIT_FAILURE);
    }
    printf("allreduce_sample executored suc \n");

    HCCLCHECK(HcclCommDestroy(hcom));
    ACLCHECK(aclrtResetDevice(device_id));
    ACLCHECK(aclFinalize());

    MPI_Finalize();
    return 0;
}
```

