RCCL Documentation

Release 0.8

Advanced Micro Devices

Jan 17, 2022

CONTENTS:

CHAPTER

ONE

RCCL

1.1 Introduction

The RCCL is an AMD port of NCCL.

CHAPTER

TWO

API

This section provides details of the library API

2.1 Communicator Functions

ncclResult_t **ncclGetUniqueId**(*ncclUniqueId* *uniqueId)
Generates an ID for ncclCommInitRank.

Generates an ID to be used in ncclCommInitRank. ncclGetUniqueId should be called once and the Id should be distributed to all ranks in the communicator before calling ncclCommInitRank.

Parameters **uniqueId** – **[in]** ncclUniqueId* pointer to uniqueId

ncclResult_t **ncclCommInitRank**(*ncclComm_t* *comm, int nranks, *ncclUniqueId* commId, int rank)
Creates a new communicator (multi thread/process version).

rank must be between 0 and nranks-1 and unique within a communicator clique. Each rank is associated to a CUDA device, which has to be set before calling ncclCommInitRank. ncclCommInitRank implicitly syncronizes with other ranks, so it must be called by different threads/processes or use ncclGroupStart/ncclGroupEnd.

Parameters **comm** – **[in]** ncclComm_t* communicator struct pointer

ncclResult_t **ncclCommInitAll**(*ncclComm_t* *comm, int ndev, const int *devlist)
Creates a clique of communicators (single process version).

This is a convenience function to create a single-process communicator clique. Returns an array of ndev newly initialized communicators in comm. comm should be pre-allocated with size at least ndev*sizeof(ncclComm_t). If devlist is NULL, the first ndev HIP devices are used. Order of devlist defines user-order of processors within the communicator.

ncclResult_t **ncclCommDestroy**(*ncclComm_t* comm)
Frees resources associated with communicator object, but waits for any operations that might still be running on the device.

ncclResult_t **ncclCommAbort**(*ncclComm_t* comm)
Frees resources associated with communicator object and aborts any operations that might still be running on the device.

ncclResult_t **ncclCommCount**(const *ncclComm_t* comm, int *count)
Gets the number of ranks in the communicator clique.

ncclResult_t **ncclCommCuDevice**(const *ncclComm_t* comm, int *device)
Returns the rocm device number associated with the communicator.

ncclResult_t **ncclCommUserRank**(const *ncclComm_t* comm, int *rank)
Returns the user-ordered "rank" associated with the communicator.

2.2 Collective Communication Operations

Collective communication operations must be called separately for each communicator in a communicator clique.

They return when operations have been enqueued on the hipstream.

Since they may perform inter-CPU synchronization, each call has to be done from a different thread or process, or need to use Group Semantics (see below).

ncclResult_t **ncclReduce**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclRedOp_t* op, int root, *ncclComm_t* comm, hipStream_t stream)

Reduce.

Reduces data arrays of length count in sendbuff into recvbuff using op operation. recvbuff may be NULL on all calls except for root device. root is the rank (not the CUDA device) where data will reside after the operation is complete.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclBcast**(void *buff, size_t count, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

(deprecated) Broadcast (in-place)

Copies count values from root to all other devices. root is the rank (not the CUDA device) where data resides before the operation is started.

This operation is implicitely in place.

ncclResult_t **ncclBroadcast**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Broadcast.

Copies count values from root to all other devices. root is the rank (not the HIP device) where data resides before the operation is started.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclAllReduce**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclRedOp_t* op, *ncclComm_t* comm, hipStream_t stream)

All-Reduce.

Reduces data arrays of length count in sendbuff using op operation, and leaves identical copies of result on each recvbuff.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclReduceScatter**(const void *sendbuff, void *recvbuff, size_t recvcount, *ncclDataType_t* datatype, *ncclRedOp_t* op, *ncclComm_t* comm, hipStream_t stream)

Reduce-Scatter.

Reduces data in sendbuff using op operation and leaves reduced result scattered over the devices so that recvbuff on rank i will contain the i-th block of the result. Assumes sendcount is equal to nranks*recvcount, which means that sendbuff should have a size of at least nranks*recvcount elements.

In-place operations will happen if recvbuff == sendbuff + rank * recvcount.

ncclResult_t **ncclAllGather**(const void *sendbuff, void *recvbuff, size_t sendcount, *ncclDataType_t* datatype, *ncclComm_t* comm, hipStream_t stream)

All-Gather.

Each device gathers sendcount values from other GPUs into recvbuff, receiving data from rank i at offset i*sendcount. Assumes recvcount is equal to nranks*sendcount, which means that recvbuff should have a size of at least nranks*sendcount elements.

In-place operations will happen if sendbuff == recvbuff + rank * sendcount.

ncclResult_t **ncclSend**(const void *sendbuff, size_t count, *ncclDataType_t* datatype, int peer, *ncclComm_t* comm, hipStream_t stream)

Send.

Send data from sendbuff to rank peer. Rank peer needs to call ncclRecv with the same datatype and the same count from this rank.

This operation is blocking for the GPU. If multiple ncclSend and ncclRecv operations need to progress concurrently to complete, they must be fused within a ncclGroupStart/ ncclGroupEnd section.

Warning: doxygenfunction: Cannot find function "ncclRecv" in doxygen xml output for project "RCCL" from directory: ../docBin/xml

ncclResult_t **ncclGather**(const void *sendbuff, void *recvbuff, size_t sendcount, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Receive.

Receive data from rank peer into recvbuff. Rank peer needs to call ncclSend with the same datatype and the same count to this rank.

This operation is blocking for the GPU. If multiple ncclSend and ncclRecv operations need to progress concurrently to complete, they must be fused within a ncclGroupStart/ ncclGroupEnd section.

Gather

Root device gathers sendcount values from other GPUs into recvbuff, receiving data from rank i at offset i*sendcount.

Assumes recvcount is equal to nranks*sendcount, which means that recvbuff should have a size of at least nranks*sendcount elements.

In-place operations will happen if sendbuff == recvbuff + rank * sendcount.

ncclResult_t **ncclScatter**(const void *sendbuff, void *recvbuff, size_t recvcount, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Scatter.

Scattered over the devices so that recvbuff on rank i will contain the i-th block of the data on root.

Assumes sendcount is equal to nranks*recvcount, which means that sendbuff should have a size of at least nranks*recvcount elements.

In-place operations will happen if recvbuff == sendbuff + rank * recvcount.

ncclResult_t **ncclAllToAll**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclComm_t* comm, hipStream_t stream)

All-To-All.

Device (i) send (j)th block of data to device (j) and be placed as (i)th block. Each block for sending/receiving has count elements, which means that recvbuff and sendbuff should have a size of nranks*count elements.

In-place operation will happen if sendbuff == recvbuff.

2.3 Group Semantics

When managing multiple GPUs from a single thread, and since NCCL collective calls may perform inter-CPU synchronization, we need to "group" calls for different ranks/devices into a single call.

Grouping NCCL calls as being part of the same collective operation is done using ncclGroupStart and ncclGroupEnd. ncclGroupStart will enqueue all collective calls until the ncclGroupEnd call, which will wait for all calls to be complete. Note that for collective communication, ncclGroupEnd only guarantees that the operations are enqueued on the streams, not that the operation is effectively done.

Both collective communication and ncclCommInitRank can be used in conjunction of ncclGroupStart/ncclGroupEnd.

ncclResult_t **ncclGroupStart**()
Group Start.

Start a group call. All calls to NCCL until ncclGroupEnd will be fused into a single NCCL operation. Nothing will be started on the CUDA stream until ncclGroupEnd.

ncclResult_t **ncclGroupEnd**()
Group End.

End a group call. Start a fused NCCL operation consisting of all calls since ncclGroupStart. Operations on the CUDA stream depending on the NCCL operations need to be called after ncclGroupEnd.

2.4 Library Functions

ncclResult_t **ncclGetVersion**(int *version)
Return the NCCL_VERSION_CODE of the NCCL library in the supplied integer.

This integer is coded with the MAJOR, MINOR and PATCH level of the NCCL library

const char ***ncclGetErrorString**(*ncclResult_t* result)
Returns a human-readable error message.

2.5 Types

There are few data structures that are internal to the library. The pointer types to these structures are given below. The user would need to use these types to create handles and pass them between different library functions.

typedef struct ncclComm ***ncclComm_t**
Opaque handle to communicator.

struct **ncclUniqueId**

2.6 Enumerations

This section provides all the enumerations used.

enum **ncclResult_t**
Error type.

Values:

enumerator **ncclSuccess**

enumerator **ncclUnhandledCudaError**

enumerator **ncclSystemError**

enumerator **ncclInternalError**

enumerator **ncclInvalidArgument**

enumerator **ncclInvalidUsage**

enumerator **ncclNumResults**

enum **ncclRedOp_t**
Values:

enumerator **ncclSum**

enumerator **ncclProd**

enumerator **ncclMax**

enumerator **ncclMin**

enumerator **ncclAvg**

enumerator **ncclNumOps**

enumerator **ncclMaxRedOp**

enum **ncclDataType_t**
Data types.

Values:

enumerator **ncclInt8**

enumerator **ncclChar**

enumerator **ncclUint8**

enumerator **ncclInt32**

enumerator **ncclInt**

enumerator **ncclUint32**

enumerator **ncclInt64**

enumerator **ncclUint64**

enumerator **ncclFloat16**

enumerator **ncclHalf**

enumerator **ncclFloat32**

enumerator **ncclFloat**

enumerator **ncclFloat64**

enumerator **ncclDouble**

enumerator **ncclBfloat16**

enumerator **ncclNumTypes**

CHAPTER

THREE

ALL API

struct **ncclUniqueId**

Public Members

char **internal**[NCCL_UNIQUE_ID_BYTES]

file **nccl.h**

#include <hip/hip_runtime.h>#include <hip/hip_fp16.h>

Defines

NCCL_MAJOR

NCCL_MINOR

NCCL_PATCH

NCCL_SUFFIX

NCCL_VERSION_CODE

NCCL_VERSION(X, Y, Z)

RCCL_BFLOAT16

RCCL_GATHER_SCATTER

RCCL_ALLTOALLV

NCCL_UNIQUE_ID_BYTES

Typedefs

typedef struct ncclComm *ncclComm_t
Opaque handle to communicator.

Enums

enum ncclResult_t
Error type.

Values:

enumerator ncclSuccess

enumerator ncclUnhandledCudaError

enumerator ncclSystemError

enumerator ncclInternalError

enumerator ncclInvalidArgument

enumerator ncclInvalidUsage

enumerator ncclNumResults

enum ncclRedOp_dummy_t
Reduction operation selector.

Values:

enumerator ncclNumOps_dummy

enum ncclRedOp_t
Values:

enumerator **ncclSum**

enumerator **ncclProd**

enumerator **ncclMax**

enumerator **ncclMin**

enumerator **ncclAvg**

enumerator **ncclNumOps**

enumerator **ncclMaxRedOp**

enum **ncclDataType_t**

Data types.

Values:

enumerator **ncclInt8**

enumerator **ncclChar**

enumerator **ncclUint8**

enumerator **ncclInt32**

enumerator **ncclInt**

enumerator **ncclUint32**

enumerator **ncclInt64**

enumerator **ncclUint64**

enumerator **ncclFloat16**

enumerator **ncclHalf**

enumerator **ncclFloat32**

enumerator **ncclFloat**

enumerator **ncclFloat64**

enumerator **ncclDouble**

enumerator **ncclBfloat16**

enumerator **ncclNumTypes**

enum **ncclScalarResidence_t**
Values:

enumerator **ncclScalarDevice**

enumerator **ncclScalarHostImmediate**

Functions

ncclResult_t **ncclGetVersion**(int *version)
Return the NCCL_VERSION_CODE of the NCCL library in the supplied integer.

This integer is coded with the MAJOR, MINOR and PATCH level of the NCCL library

ncclResult_t **ncclGetUniqueId**(*ncclUniqueId* *uniqueId)
Generates an ID for ncclCommInitRank.

Generates an ID to be used in ncclCommInitRank. ncclGetUniqueId should be called once and the Id should be distributed to all ranks in the communicator before calling ncclCommInitRank.

Parameters **uniqueId** – **[in]** ncclUniqueId* pointer to uniqueId

ncclResult_t **ncclCommInitRank**(*ncclComm_t* *comm, int nranks, *ncclUniqueId* commId, int rank)
Creates a new communicator (multi thread/process version).

rank must be between 0 and nranks-1 and unique within a communicator clique. Each rank is associated to a CUDA device, which has to be set before calling ncclCommInitRank. ncclCommInitRank implic-

itly syncronizes with other ranks, so it must be called by different threads/processes or use ncclGroupStart/ncclGroupEnd.

Parameters **comm** – **[in]** ncclComm_t* communicator struct pointer

ncclResult_t **ncclCommInitAll**(*ncclComm_t* *comm, int ndev, const int *devlist)
Creates a clique of communicators (single process version).

This is a convenience function to create a single-process communicator clique. Returns an array of ndev newly initialized communicators in comm. comm should be pre-allocated with size at least ndev*sizeof(ncclComm_t). If devlist is NULL, the first ndev HIP devices are used. Order of devlist defines user-order of processors within the communicator.

ncclResult_t **ncclCommDestroy**(*ncclComm_t* comm)
Frees resources associated with communicator object, but waits for any operations that might still be running on the device.

ncclResult_t **ncclCommAbort**(*ncclComm_t* comm)
Frees resources associated with communicator object and aborts any operations that might still be running on the device.

const char ***ncclGetErrorString**(*ncclResult_t* result)
Returns a human-readable error message.

const char ***pncclGetErrorString**(*ncclResult_t* result)

ncclResult_t **ncclCommGetAsyncError**(*ncclComm_t* comm, *ncclResult_t* *asyncError)
Checks whether the comm has encountered any asynchronous errors.

ncclResult_t **ncclCommCount**(const *ncclComm_t* comm, int *count)
Gets the number of ranks in the communicator clique.

ncclResult_t **ncclCommCuDevice**(const *ncclComm_t* comm, int *device)
Returns the rocm device number associated with the communicator.

ncclResult_t **ncclCommUserRank**(const *ncclComm_t* comm, int *rank)
Returns the user-ordered "rank" associated with the communicator.

ncclResult_t **ncclRedOpCreatePreMulSum**(*ncclRedOp_t* *op, void *scalar, *ncclDataType_t* datatype, *ncclScalarResidence_t* residence, *ncclComm_t* comm)

ncclResult_t **pncclRedOpCreatePreMulSum**(*ncclRedOp_t* *op, void *scalar, *ncclDataType_t* datatype, *ncclScalarResidence_t* residence, *ncclComm_t* comm)

ncclResult_t **ncclRedOpDestroy**(*ncclRedOp_t* op, *ncclComm_t* comm)

ncclResult_t **pncclRedOpDestroy**(*ncclRedOp_t* op, *ncclComm_t* comm)

ncclResult_t **ncclReduce**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclRedOp_t* op, int root, *ncclComm_t* comm, hipStream_t stream)
Reduce.

Reduces data arrays of length count in sendbuff into recvbuff using op operation. recvbuff may be NULL on all calls except for root device. root is the rank (not the CUDA device) where data will reside after the operation is complete.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclBcast**(void *buff, size_t count, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

(deprecated) Broadcast (in-place)

Copies count values from root to all other devices. root is the rank (not the CUDA device) where data resides before the operation is started.

This operation is implicitely in place.

ncclResult_t **ncclBroadcast**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Broadcast.

Copies count values from root to all other devices. root is the rank (not the HIP device) where data resides before the operation is started.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclAllReduce**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclRedOp_t* op, *ncclComm_t* comm, hipStream_t stream)

All-Reduce.

Reduces data arrays of length count in sendbuff using op operation, and leaves identical copies of result on each recvbuff.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclReduceScatter**(const void *sendbuff, void *recvbuff, size_t recvcount, *ncclDataType_t* datatype, *ncclRedOp_t* op, *ncclComm_t* comm, hipStream_t stream)

Reduce-Scatter.

Reduces data in sendbuff using op operation and leaves reduced result scattered over the devices so that recvbuff on rank i will contain the i-th block of the result. Assumes sendcount is equal to nranks*recvcount, which means that sendbuff should have a size of at least nranks*recvcount elements.

In-place operations will happen if recvbuff == sendbuff + rank * recvcount.

ncclResult_t **ncclAllGather**(const void *sendbuff, void *recvbuff, size_t sendcount, *ncclDataType_t* datatype, *ncclComm_t* comm, hipStream_t stream)

All-Gather.

Each device gathers sendcount values from other GPUs into recvbuff, receiving data from rank i at offset i*sendcount. Assumes recvcount is equal to nranks*sendcount, which means that recvbuff should have a size of at least nranks*sendcount elements.

In-place operations will happen if sendbuff == recvbuff + rank * sendcount.

ncclResult_t **ncclSend**(const void *sendbuff, size_t count, *ncclDataType_t* datatype, int peer, *ncclComm_t* comm, hipStream_t stream)

Send.

Send data from sendbuff to rank peer. Rank peer needs to call ncclRecv with the same datatype and the same count from this rank.

This operation is blocking for the GPU. If multiple ncclSend and ncclRecv operations need to progress concurrently to complete, they must be fused within a ncclGroupStart/ ncclGroupEnd section.

ncclResult_t **ncclGather**(const void *sendbuff, void *recvbuff, size_t sendcount, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Receive.

Receive data from rank peer into recvbuff. Rank peer needs to call ncclSend with the same datatype and the same count to this rank.

This operation is blocking for the GPU. If multiple ncclSend and ncclRecv operations need to progress concurrently to complete, they must be fused within a ncclGroupStart/ ncclGroupEnd section.

Gather

Root device gathers sendcount values from other GPUs into recvbuff, receiving data from rank i at offset i*sendcount.

Assumes recvcount is equal to nranks*sendcount, which means that recvbuff should have a size of at least nranks*sendcount elements.

In-place operations will happen if sendbuff == recvbuff + rank * sendcount.

ncclResult_t **ncclScatter**(const void *sendbuff, void *recvbuff, size_t recvcount, *ncclDataType_t* datatype, int root, *ncclComm_t* comm, hipStream_t stream)

Scatter.

Scattered over the devices so that recvbuff on rank i will contain the i-th block of the data on root.

Assumes sendcount is equal to nranks*recvcount, which means that sendbuff should have a size of at least nranks*recvcount elements.

In-place operations will happen if recvbuff == sendbuff + rank * recvcount.

ncclResult_t **ncclAllToAll**(const void *sendbuff, void *recvbuff, size_t count, *ncclDataType_t* datatype, *ncclComm_t* comm, hipStream_t stream)

All-To-All.

Device (i) send (j)th block of data to device (j) and be placed as (i)th block. Each block for sending/receiving has count elements, which means that recvbuff and sendbuff should have a size of nranks*count elements.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclAllToAllv**(const void *sendbuff, const size_t sendcounts[], const size_t sdispls[], void *recvbuff, const size_t recvcounts[], const size_t rdispls[], *ncclDataType_t* datatype, *ncclComm_t* comm, hipStream_t stream)

All-To-Allv.

Device (i) sends sendcounts[j] of data from offset sdispls[j] to device (j). In the same time, device (i) receives recvcounts[j] of data from device (j) to be placed at rdispls[j].

sendcounts, sdispls, recvcounts and rdispls are all measured in the units of datatype, not bytes.

In-place operation will happen if sendbuff == recvbuff.

ncclResult_t **ncclGroupStart**()

Group Start.

Start a group call. All calls to NCCL until ncclGroupEnd will be fused into a single NCCL operation. Nothing will be started on the CUDA stream until ncclGroupEnd.

ncclResult_t **ncclGroupEnd**()

Group End.

End a group call. Start a fused NCCL operation consisting of all calls since ncclGroupStart. Operations on the CUDA stream depending on the NCCL operations need to be called after ncclGroupEnd.

CHAPTER

FOUR

ATTRIBUTIONS

Contains contributions from NVIDIA.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of NVIDIA CORPORATION, Lawrence Berkeley National Laboratory, the U.S. Department of Energy, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The U.S. Department of Energy funded the development of this software under subcontract 7078610 with Lawrence Berkeley National Laboratory.

This code also includes files from the NVIDIA Tools Extension SDK project.

For more information and license details, see

https://github.com/NVIDIA/NVTX

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

INDEX

N

P

R