RCCL Documentation

Release 0.8

Advanced Mirco Devices

Jul 02, 2020

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

- `[in] uniqueId`: 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

- `[in] comm`: 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.

ncclResult_t **ncclRecv**(void **recvbuff*, size_t *count*, *ncclDataType_t datatype*, int *peer*, *ncclComm_t comm*, hipStream_t *stream*)

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

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 = 0

enumerator ncclUnhandledCudaError = 1

enumerator ncclSystemError = 2

enumerator ncclInternalError = 3

enumerator ncclInvalidArgument = 4

enumerator ncclInvalidUsage = 5

enumerator ncclNumResults = 6

enum ncclRedOp_t
Reduction operation selector.

Values:

enumerator ncclSum = 0

enumerator ncclProd = 1

enumerator ncclMax = 2

enumerator ncclMin = 3

enumerator ncclNumOps = 4

enum ncclDataType_t
Data types.

Values:

enumerator ncclInt8 = 0

enumerator ncclChar = 0

enumerator ncclUint8 = 1

enumerator ncclInt32 = 2

enumerator ncclInt = 2

enumerator ncclUint32 = 3

enumerator ncclInt64 = 4

enumerator ncclUint64 = 5

enumerator ncclFloat16 = 6

enumerator ncclHalf = 6

enumerator ncclFloat32 = 7

enumerator ncclFloat = 7

enumerator ncclFloat64 = 8

enumerator ncclDouble = 8

enumerator ncclBfloat16 = 9

enumerator ncclNumTypes = 10

CHAPTER

THREE

ALL API

struct ncclUniqueId

Public Members

char **internal**[**NCCL_UNIQUE_ID_BYTES**]

file **nccl.h**

#include <hip/hip_runtime_api.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

NCCL_UNIQUE_ID_BYTES

Typedefs

typedef struct ncclComm ***ncclComm_t**

Opaque handle to communicator.

Enums

enum ncclResult_t
Error type.

Values:

enumerator ncclSuccess = 0

enumerator ncclUnhandledCudaError = 1

enumerator ncclSystemError = 2

enumerator ncclInternalError = 3

enumerator ncclInvalidArgument = 4

enumerator ncclInvalidUsage = 5

enumerator ncclNumResults = 6

enum ncclRedOp_t
Reduction operation selector.

Values:

enumerator ncclSum = 0

enumerator ncclProd = 1

enumerator ncclMax = 2

enumerator ncclMin = 3

enumerator ncclNumOps = 4

enum ncclDataType_t
Data types.

Values:

enumerator ncclInt8 = 0

enumerator ncclChar = 0

enumerator ncclUint8 = 1

enumerator ncclInt32 = 2

enumerator ncclInt = 2

enumerator ncclUint32 = 3

enumerator ncclInt64 = 4

enumerator ncclUint64 = 5

enumerator ncclFloat16 = 6

enumerator ncclHalf = 6

enumerator ncclFloat32 = 7

enumerator ncclFloat = 7

enumerator ncclFloat64 = 8

enumerator ncclDouble = 8

enumerator ncclBfloat16 = 9

enumerator ncclNumTypes = 10

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 **pncclGetVersion** (int **version*)

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

- `[in] uniqueId`: ncclUniqueId* pointer to uniqueId

ncclResult_t **pncclGetUniqueId** (*ncclUniqueId* **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

- `[in] comm`: ncclComm_t* communicator struct pointer

ncclResult_t **pncclCommInitRank** (*ncclComm_t* **comm*, int *nranks*, *ncclUniqueId* *commId*, int *rank*)

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 **pncclCommInitAll** (*ncclComm_t* **comm*, int *ndev*, **const** int **devlist*)

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 **pncclCommDestroy** (*ncclComm_t* *comm*)

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 **pncclCommAbort** (*ncclComm_t* *comm*)

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 **pncclCommGetAsyncError**(*ncclComm_t comm*, *ncclResult_t *asyncError*)

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

ncclResult_t **pncclCommCount**(**const** *ncclComm_t comm*, int **count*)

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

ncclResult_t **pncclCommCuDevice**(**const** *ncclComm_t comm*, int **device*)

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

ncclResult_t **pncclCommUserRank**(**const** *ncclComm_t comm*, int **rank*)

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

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

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

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

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

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

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

ncclResult_t **pncclRecv**(void **recvbuff*, size_t *count*, *ncclDataType_t datatype*, int *peer*, *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.

ncclResult_t **ncclRecv**(void **recvbuff*, size_t *count*, *ncclDataType_t datatype*, int *peer*, *ncclComm_t comm*, hipStream_t *stream*)

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

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

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

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

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 **pncclGroupStart**()

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.

ncclResult_t **pncclGroupEnd**()

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

INDEX

N

P

R