RCCL Documentation

Release 2.20.5

Advanced Micro Devices, Inc.

Nov 19, 2024

API REFERENCE

RCCL (pronounced "Rickel") is a stand-alone library that provides multi-GPU and multi-node collective communication primitives optimized for AMD GPUs. It implements routines such as `all-reduce`, `all-gather`, `reduce`, `broadcast`, `reduce-scatter`, `gather`, `scatter`, `all-to-allv`, and `all-to-all` as well as direct point-to-point (GPU-to-GPU) send and receive operations. The provided collective communication routines are implemented using Ring and Tree algorithms. They are optimized to achieve high bandwidth and low latency by leveraging topology awareness, high-speed interconnects, and RDMA based collectives.

RCCL utilizes PCIe and xGMI high-speed interconnects for intra-node communication as well as InfiniBand, RoCE, and TCP/IP for inter-node communication. It supports an arbitrary number of GPUs installed in a single-node or multi-node platform and can be easily integrated into single- or multi-process (e.g., MPI) applications.

You can access RCCL code on our GitHub repository.

The documentation is structured as follows:

API reference

- *Library specification*
- *API library*

To contribute to the documentation refer to Contributing to ROCm.

Licensing information can be found on the Licensing page.

CHAPTER

ONE

RCCL LIBRARY SPECIFICATION

This document provides details of the API library.

1.1 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 by a single rank and the ID should be distributed to all ranks in the communicator before using it as a parameter for ncclCommInitRank.

Parameters
uniqueId – **[out]** Pointer to where uniqueId will be stored

Returns
Result code. See *Result Codes* for more details.

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** – **[out]** Pointer to created communicator
- **nranks** – **[in]** Total number of ranks participating in this communicator
- **commId** – **[in]** UniqueId required for initialization
- **rank** – **[in]** Current rank to create communicator for

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **comm** – **[out]** Pointer to array of created communicators
- **ndev** – **[in]** Total number of ranks participating in this communicator
- **devlist** – **[in]** Array of GPU device indices to create for

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommDestroy**(*ncclComm_t* comm)

Frees local resources associated with communicator object.

Destroy all local resources associated with the passed in communicator object

Parameters
comm – **[in]** Communicator to destroy

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommAbort**(*ncclComm_t* comm)

Abort any in-progress calls and destroy the communicator object.

Frees resources associated with communicator object and aborts any operations that might still be running on the device.

Parameters
comm – **[in]** Communicator to abort and destroy

Returns
Result code. See *Result Codes* for more details.

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

Gets the number of ranks in the communicator clique.

Returns the number of ranks in the communicator clique (as set during initialization)

Parameters

- **comm** – **[in]** Communicator to query
- **count** – **[out]** Pointer to where number of ranks will be stored

Returns
Result code. See *Result Codes* for more details.

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

Get the ROCm device index associated with a communicator.

Returns the ROCm device number associated with the provided communicator.

Parameters

- **comm** – **[in]** Communicator to query
- **device** – **[out]** Pointer to where the associated ROCm device index will be stored

Returns
Result code. See *Result Codes* for more details.

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

Get the rank associated with a communicator.

Returns the user-ordered "rank" associated with the provided communicator.

Parameters

- **comm** – **[in]** Communicator to query
- **rank** – **[out]** Pointer to where the associated rank will be stored

Returns
Result code. See *Result Codes* for more details.

1.2 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 HIP device) where data will reside after the operation is complete. In-place operation will happen if sendbuff == recvbuff.

Parameters

- **sendbuff** – **[in]** Local device data buffer to be reduced
- **recvbuff** – **[out]** Data buffer where result is stored (only for *root* rank). May be null for other ranks.
- **count** – **[in]** Number of elements in every send buffer
- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator type
- **root** – **[in]** Rank where result data array will be stored
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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 implicitly in-place.

Parameters

- **buff** – **[inout]** Input array on *root* to be copied to other ranks. Output array for all ranks.
- **count** – **[in]** Number of elements in data buffer
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank owning buffer to be copied to others
- **comm** – **[in]** Communicator group object to execute on

- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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 *sendbuff* on *root* to *recvbuff* on all devices. root* is the rank (not the HIP device) where data resides before the operation is started. sendbuff* may be NULL on ranks other than *root*. In-place operation will happen if *sendbuff* == *recvbuff*.

Parameters

- **sendbuff** – **[in]** Data array to copy (if *root*). May be NULL for other ranks
- **recvbuff** – **[in]** Data array to store received array
- **count** – **[in]** Number of elements in data buffer
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank of broadcast root
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to reduce
- **recvbuff** – **[out]** Data array to store reduced result array
- **count** – **[in]** Number of elements in data buffer
- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to reduce
- **recvbuff** – **[out]** Data array to store reduced result subarray
- **recvcount** – **[in]** Number of elements each rank receives
- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to send
- **recvbuff** – **[out]** Data array to store the gathered result
- **sendcount** – **[in]** Number of elements each rank sends
- **datatype** – **[in]** Data buffer element datatype
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

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* as 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.

Parameters

- **sendbuff** – **[in]** Data array to send
- **count** – **[in]** Number of elements to send
- **datatype** – **[in]** Data buffer element datatype
- **peer** – **[in]** Peer rank to send to
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclRecv**(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 as 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.

Parameters

- **recvbuff** – **[out]** Data array to receive
- **count** – **[in]** Number of elements to receive
- **datatype** – **[in]** Data buffer element datatype
- **peer** – **[in]** Peer rank to send to
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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. recvbuff* may be NULL on ranks other than *root*.

Parameters

- **sendbuff** – **[in]** Data array to send
- **recvbuff** – **[out]** Data array to receive into on *root*.
- **sendcount** – **[in]** Number of elements to send per rank
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank that receives data from all other ranks
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Data array to send (on *root* rank). May be NULL on other ranks.
- **recvbuff** – **[out]** Data array to receive partial subarray into
- **recvcount** – **[in]** Number of elements to receive per rank
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank that scatters data to all other ranks
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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 is NOT supported. It is the user's responsibility to ensure that sendbuff and recvbuff are distinct.

Parameters

- **sendbuff** – **[in]** Data array to send (contains blocks for each other rank)
- **recvbuff** – **[out]** Data array to receive (contains blocks from each other rank)
- **count** – **[in]** Number of elements to send between each pair of ranks
- **datatype** – **[in]** Data buffer element datatype
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

1.3 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 RCCL until ncclGroupEnd will be fused into a single RCCL operation. Nothing will be started on the HIP stream until ncclGroupEnd.

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclGroupEnd**()

Group End.

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

Returns
Result code. See *Result Codes* for more details.

1.4 Library functions

ncclResult_t **ncclGetVersion**(int *version)

Return the RCCL_VERSION_CODE of RCCL in the supplied integer.

This integer is coded with the MAJOR, MINOR and PATCH level of RCCL.

Parameters
version – **[out]** Pointer to where version will be stored

Returns
Result code. See *Result Codes* for more details.

const char ***ncclGetErrorString**(*ncclResult_t* result)

Returns a string for each result code.

Returns a human-readable string describing the given result code.

Parameters
result – **[in]** Result code to get description for

Returns
String containing description of result code.

1.5 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.

A communicator contains information required to facilitate collective communications calls

struct **ncclUniqueId**

Opaque unique id used to initialize communicators.

The *ncclUniqueId* must be passed to all participating ranks

1.6 Enumerations

This section provides all the enumerations used.

enum **ncclResult_t**

Result type.

Return codes aside from ncclSuccess indicate that a call has failed

Values:

enumerator **ncclSuccess**

No error

enumerator **ncclUnhandledCudaError**

Unhandled HIP error

enumerator **ncclSystemError**

Unhandled system error

enumerator **ncclInternalError**

Internal Error - Please report to RCCL developers

enumerator **ncclInvalidArgument**

Invalid argument

enumerator **ncclInvalidUsage**

Invalid usage

enumerator **ncclRemoteError**

Remote process exited or there was a network error

enumerator **ncclInProgress**

RCCL operation in progress

enumerator **ncclNumResults**

Number of result types

enum **ncclRedOp_t**

Reduction operation selector.

Enumeration used to specify the various reduction operations ncclNumOps is the number of built-in ncclRedOp_t values and serves as the least possible value for dynamic ncclRedOp_t values constructed by ncclRedOpCreate functions.

ncclMaxRedOp is the largest valid value for ncclRedOp_t and is defined to be the largest signed value (since compilers are permitted to use signed enums) that won't grow sizeof(ncclRedOp_t) when compared to previous RCCL versions to maintain ABI compatibility.

Values:

enumerator **ncclSum**

Sum

enumerator **ncclProd**

Product

enumerator **ncclMax**

Max

enumerator **ncclMin**

Min

enumerator **ncclAvg**

Average

enumerator **ncclNumOps**

Number of built-in reduction ops

enumerator **ncclMaxRedOp**

Largest value for ncclRedOp_t

enum **ncclDataType_t**

Data types.

Enumeration of the various supported datatype

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 **ncclFp8E4M3**

enumerator **ncclFp8E5M2**

enumerator **ncclNumTypes**

CHAPTER

TWO

API LIBRARY

struct **ncclConfig_t**

Communicator configuration.

Users can assign value to attributes to specify the behavior of a communicator

Public Members

size_t **size**

Should not be touched

unsigned int **magic**

Should not be touched

unsigned int **version**

Should not be touched

int **blocking**

Whether or not calls should block or not

int **cgaClusterSize**

Cooperative group array cluster size

int **minCTAs**

Minimum number of cooperative thread arrays (blocks)

int **maxCTAs**

Maximum number of cooperative thread arrays (blocks)

const char ***netName**

Force NCCL to use a specfic network

int **splitShare**

Allow communicators to share resources

struct **ncclUniqueId**

Opaque unique id used to initialize communicators.

The *ncclUniqueId* must be passed to all participating ranks

Public Members

char **internal**[NCCL_UNIQUE_ID_BYTES]

Opaque array>

file **mainpage.txt**

file **nccl.h.in**

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

Defines

NCCL_H_

NCCL_MAJOR

NCCL_MINOR

NCCL_PATCH

NCCL_SUFFIX

NCCL_VERSION_CODE

NCCL_VERSION(X, Y, Z)

RCCL_BFLOAT16

RCCL_FLOAT8

RCCL_GATHER_SCATTER

RCCL_ALLTOALLV

NCCL_COMM_NULL

NCCL_UNIQUE_ID_BYTES

NCCL_CONFIG_UNDEF_INT

NCCL_CONFIG_UNDEF_PTR

NCCL_SPLIT_NOCOLOR

NCCL_CONFIG_INITIALIZER

Typedefs

typedef struct ncclComm *__ncclComm_t__

Opaque handle to communicator.

A communicator contains information required to facilitate collective communications calls

typedef int __mscclAlgoHandle_t__

Opaque handle to MSCCL algorithm.

Enums

enum __ncclResult_t__

Result type.

Return codes aside from ncclSuccess indicate that a call has failed

Values:

enumerator __ncclSuccess__

No error

enumerator __ncclUnhandledCudaError__

Unhandled HIP error

enumerator __ncclSystemError__

Unhandled system error

enumerator __ncclInternalError__

Internal Error - Please report to RCCL developers

enumerator __ncclInvalidArgument__

Invalid argument

enumerator __ncclInvalidUsage__

Invalid usage

enumerator __ncclRemoteError__

Remote process exited or there was a network error

enumerator **ncclInProgress**

RCCL operation in progress

enumerator **ncclNumResults**

Number of result types

enum **ncclRedOp_dummy_t**

Dummy reduction enumeration.

Dummy reduction enumeration used to determine value for ncclMaxRedOp

Values:

enumerator **ncclNumOps_dummy**

enum **ncclRedOp_t**

Reduction operation selector.

Enumeration used to specify the various reduction operations ncclNumOps is the number of built-in ncclRedOp_t values and serves as the least possible value for dynamic ncclRedOp_t values constructed by ncclRedOpCreate functions.

ncclMaxRedOp is the largest valid value for ncclRedOp_t and is defined to be the largest signed value (since compilers are permitted to use signed enums) that won't grow sizeof(ncclRedOp_t) when compared to previous RCCL versions to maintain ABI compatibility.

Values:

enumerator **ncclSum**

Sum

enumerator **ncclProd**

Product

enumerator **ncclMax**

Max

enumerator **ncclMin**

Min

enumerator **ncclAvg**

Average

enumerator **ncclNumOps**

Number of built-in reduction ops

enumerator **ncclMaxRedOp**

Largest value for ncclRedOp_t

enum **ncclDataType_t**

Data types.

Enumeration of the various supported datatype

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 **ncclFp8E4M3**

enumerator **ncclFp8E5M2**

enumerator **ncclNumTypes**

enum **ncclScalarResidence_t**

Location and dereferencing logic for scalar arguments.

Enumeration specifying memory location of the scalar argument. Based on where the value is stored, the argument will be dereferenced either while the collective is running (if in device memory), or before the ncclRedOpCreate() function returns (if in host memory).

Values:

enumerator **ncclScalarDevice**

Scalar is in device-visible memory

enumerator **ncclScalarHostImmediate**

Scalar is in host-visible memory

Functions

ncclResult_t **ncclMemAlloc**(void **ptr, size_t size)

ncclResult_t **pncclMemAlloc**(void **ptr, size_t size)

ncclResult_t **ncclMemFree**(void *ptr)

ncclResult_t **pncclMemFree**(void *ptr)

ncclResult_t **ncclGetVersion**(int *version)

Return the RCCL_VERSION_CODE of RCCL in the supplied integer.

This integer is coded with the MAJOR, MINOR and PATCH level of RCCL.

Parameters
version – **[out]** Pointer to where version will be stored

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclGetUniqueId**(*ncclUniqueId* *uniqueId)

Generates an ID for ncclCommInitRank.

Generates an ID to be used in ncclCommInitRank. ncclGetUniqueId should be called once by a single rank and the ID should be distributed to all ranks in the communicator before using it as a parameter for ncclCommInitRank.

Parameters
uniqueId – **[out]** Pointer to where uniqueId will be stored

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommInitRankConfig**(*ncclComm_t* *comm, int nranks, *ncclUniqueId* commId, int rank, *ncclConfig_t* *config)

Create a new communicator with config.

Create a new communicator (multi thread/process version) with a configuration set by users. See *Communicator Configuration* for more details. Each rank is associated to a CUDA device, which has to be set before calling ncclCommInitRank.

Parameters

- **comm** – **[out]** Pointer to created communicator
- **nranks** – **[in]** Total number of ranks participating in this communicator
- **commId** – **[in]** UniqueId required for initialization
- **rank** – **[in]** Current rank to create communicator for. [0 to nranks-1]
- **config** – **[in]** Pointer to communicator configuration

Returns
Result code. See *Result Codes* for more details.

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** – **[out]** Pointer to created communicator
- **nranks** – **[in]** Total number of ranks participating in this communicator
- **commId** – **[in]** UniqueId required for initialization
- **rank** – **[in]** Current rank to create communicator for

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **comm** – **[out]** Pointer to array of created communicators
- **ndev** – **[in]** Total number of ranks participating in this communicator
- **devlist** – **[in]** Array of GPU device indices to create for

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommFinalize**(*ncclComm_t* comm)

Finalize a communicator.

ncclCommFinalize flushes all issued communications and marks communicator state as ncclInProgress. The state will change to ncclSuccess when the communicator is globally quiescent and related resources are freed; then, calling ncclCommDestroy can locally free the rest of the resources (e.g. communicator itself) without blocking.

Parameters
comm – **[in]** Communicator to finalize

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommDestroy**(*ncclComm_t* comm)

Frees local resources associated with communicator object.

Destroy all local resources associated with the passed in communicator object

Parameters
comm – **[in]** Communicator to destroy

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommAbort**(*ncclComm_t* comm)

Abort any in-progress calls and destroy the communicator object.

Frees resources associated with communicator object and aborts any operations that might still be running on the device.

Parameters
comm – **[in]** Communicator to abort and destroy

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommSplit**(*ncclComm_t* comm, int color, int key, *ncclComm_t* *newcomm, *ncclConfig_t* *config)

Create one or more communicators from an existing one.

Creates one or more communicators from an existing one. Ranks with the same color will end up in the same communicator. Within the new communicator, key will be used to order ranks. NCCL_SPLIT_NOCOLOR as color will indicate the rank will not be part of any group and will therefore return a NULL communicator. If config is NULL, the new communicator will inherit the original communicator's configuration

Parameters

- **comm** – **[in]** Original communicator object for this rank
- **color** – **[in]** Color to assign this rank
- **key** – **[in]** Key used to order ranks within the same new communicator
- **newcomm** – **[out]** Pointer to new communicator
- **config** – **[in]** Config file for new communicator. May be NULL to inherit from comm

Returns
Result code. See *Result Codes* for more details.

const char ***ncclGetErrorString**(*ncclResult_t* result)

Returns a string for each result code.

Returns a human-readable string describing the given result code.

Parameters
result – **[in]** Result code to get description for

Returns
String containing description of result code.

const char ***ncclGetLastError**(*ncclComm_t* comm)

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

Checks whether the comm has encountered any asynchronous errors.

Query whether the provided communicator has encountered any asynchronous errors

Parameters

- **comm** – **[in]** Communicator to query
- **asyncError** – **[out]** Pointer to where result code will be stored

Returns
Result code. See *Result Codes* for more details.

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

Gets the number of ranks in the communicator clique.

Returns the number of ranks in the communicator clique (as set during initialization)

Parameters

- **comm** – **[in]** Communicator to query
- **count** – **[out]** Pointer to where number of ranks will be stored

Returns
Result code. See *Result Codes* for more details.

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

Get the ROCm device index associated with a communicator.

Returns the ROCm device number associated with the provided communicator.

Parameters

- **comm** – **[in]** Communicator to query
- **device** – **[out]** Pointer to where the associated ROCm device index will be stored

Returns
Result code. See *Result Codes* for more details.

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

Get the rank associated with a communicator.

Returns the user-ordered "rank" associated with the provided communicator.

Parameters

- **comm** – **[in]** Communicator to query
- **rank** – **[out]** Pointer to where the associated rank will be stored

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclCommRegister**(const *ncclComm_t* comm, void *buff, size_t size, void **handle)

ncclResult_t **ncclCommDeregister**(const *ncclComm_t* comm, void *handle)

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

Create a custom pre-multiplier reduction operator.

Creates a new reduction operator which pre-multiplies input values by a given scalar locally before reducing them with peer values via summation. For use only with collectives launched against *comm* and *datatype*.

The residence* argument indicates how/when the memory pointed to by *scalar* will be dereferenced. Upon return, the newly created operator's handle is stored in *op*.

Parameters

- **op** – **[out]** Pointer to where newly created custom reduction operator is to be stored
- **scalar** – **[in]** Pointer to scalar value.
- **datatype** – **[in]** Scalar value datatype
- **residence** – **[in]** Memory type of the scalar value
- **comm** – **[in]** Communicator to associate with this custom reduction operator

Returns

Result code. See *Result Codes* for more details.

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

Destroy custom reduction operator.

Destroys the reduction operator *op*. The operator must have been created by ncclRedOpCreatePreMul with the matching communicator *comm*. An operator may be destroyed as soon as the last RCCL function which is given that operator returns.

Parameters

- **op** – **[in]** Custom reduction operator is to be destroyed
- **comm** – **[in]** Communicator associated with this reduction operator

Returns

Result code. See *Result Codes* for more details.

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 HIP device) where data will reside after the operation is complete. In-place operation will happen if sendbuff == recvbuff.

Parameters

- **sendbuff** – **[in]** Local device data buffer to be reduced
- **recvbuff** – **[out]** Data buffer where result is stored (only for *root* rank). May be null for other ranks.
- **count** – **[in]** Number of elements in every send buffer
- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator type
- **root** – **[in]** Rank where result data array will be stored
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

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 implicitly in-place.

Parameters

- **buff** – **[inout]** Input array on *root* to be copied to other ranks. Output array for all ranks.
- **count** – **[in]** Number of elements in data buffer
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank owning buffer to be copied to others
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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 *sendbuff* on *root* to *recvbuff* on all devices. root* is the rank (not the HIP device) where data resides before the operation is started. sendbuff* may be NULL on ranks other than *root*. In-place operation will happen if *sendbuff* == *recvbuff*.

Parameters

- **sendbuff** – **[in]** Data array to copy (if *root*). May be NULL for other ranks
- **recvbuff** – **[in]** Data array to store received array
- **count** – **[in]** Number of elements in data buffer
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank of broadcast root
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to reduce
- **recvbuff** – **[out]** Data array to store reduced result array
- **count** – **[in]** Number of elements in data buffer

- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to reduce
- **recvbuff** – **[out]** Data array to store reduced result subarray
- **recvcount** – **[in]** Number of elements each rank receives
- **datatype** – **[in]** Data buffer element datatype
- **op** – **[in]** Reduction operator
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Input data array to send
- **recvbuff** – **[out]** Data array to store the gathered result
- **sendcount** – **[in]** Number of elements each rank sends
- **datatype** – **[in]** Data buffer element datatype
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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* as 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.

Parameters

- **sendbuff** – **[in]** Data array to send
- **count** – **[in]** Number of elements to send
- **datatype** – **[in]** Data buffer element datatype
- **peer** – **[in]** Peer rank to send to
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

ncclResult_t **ncclRecv**(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 as 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.

Parameters

- **recvbuff** – **[out]** Data array to receive
- **count** – **[in]** Number of elements to receive
- **datatype** – **[in]** Data buffer element datatype
- **peer** – **[in]** Peer rank to send to
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

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. recvbuff* may be NULL on ranks other than *root*.

Parameters

- **sendbuff** – **[in]** Data array to send

- **recvbuff** – **[out]** Data array to receive into on *root*.
- **sendcount** – **[in]** Number of elements to send per rank
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank that receives data from all other ranks
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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.

Parameters

- **sendbuff** – **[in]** Data array to send (on *root* rank). May be NULL on other ranks.
- **recvbuff** – **[out]** Data array to receive partial subarray into
- **recvcount** – **[in]** Number of elements to receive per rank
- **datatype** – **[in]** Data buffer element datatype
- **root** – **[in]** Rank that scatters data to all other ranks
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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 is NOT supported. It is the user's responsibility to ensure that sendbuff and recvbuff are distinct.

Parameters

- **sendbuff** – **[in]** Data array to send (contains blocks for each other rank)
- **recvbuff** – **[out]** Data array to receive (contains blocks from each other rank)
- **count** – **[in]** Number of elements to send between each pair of ranks
- **datatype** – **[in]** Data buffer element datatype
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

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). At 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.

Parameters

- **sendbuff** – **[in]** Data array to send (contains blocks for each other rank)
- **sendcounts** – **[in]** Array containing number of elements to send to each participating rank
- **sdispls** – **[in]** Array of offsets into *sendbuff* for each participating rank
- **recvbuff** – **[out]** Data array to receive (contains blocks from each other rank)
- **recvcounts** – **[in]** Array containing number of elements to receive from each participating rank
- **rdispls** – **[in]** Array of offsets into *recvbuff* for each participating rank
- **datatype** – **[in]** Data buffer element datatype
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns

Result code. See *Result Codes* for more details.

ncclResult_t **mscclLoadAlgo**(const char *mscclAlgoFilePath, *mscclAlgoHandle_t* *mscclAlgoHandle, int rank)

MSCCL Load Algorithm.

Load MSCCL algorithm file specified in mscclAlgoFilePath and return its handle via mscclAlgoHandle. This API is expected to be called by MSCCL scheduler instead of end users.

Parameters

- **mscclAlgoFilePath** – **[in]** Path to MSCCL algorithm file
- **mscclAlgoHandle** – **[out]** Returned handle to MSCCL algorithm
- **rank** – **[in]** Current rank

Returns

Result code. See *Result Codes* for more details.

ncclResult_t **mscclRunAlgo**(const void *sendBuff, const size_t sendCounts[], const size_t sDisPls[], void *recvBuff, const size_t recvCounts[], const size_t rDisPls[], size_t count, *ncclDataType_t* dataType, int root, int peer, *ncclRedOp_t* op, *mscclAlgoHandle_t* mscclAlgoHandle, *ncclComm_t* comm, hipStream_t stream)

MSCCL Run Algorithm.

Run MSCCL algorithm specified by mscclAlgoHandle. The parameter list merges all possible parameters required by different operations as this is a general-purposed API. This API is expected to be called by MSCCL scheduler instead of end users.

Parameters

- **sendBuff** – **[in]** Data array to send
- **sendCounts** – **[in]** Array containing number of elements to send to each participating rank
- **sDisPls** – **[in]** Array of offsets into *sendbuff* for each participating rank
- **recvBuff** – **[out]** Data array to receive
- **recvCounts** – **[in]** Array containing number of elements to receive from each participating rank
- **rDisPls** – **[in]** Array of offsets into *recvbuff* for each participating rank
- **count** – **[in]** Number of elements
- **dataType** – **[in]** Data buffer element datatype
- **root** – **[in]** Root rank index
- **peer** – **[in]** Peer rank index
- **op** – **[in]** Reduction operator
- **mscclAlgoHandle** – **[in]** Handle to MSCCL algorithm
- **comm** – **[in]** Communicator group object to execute on
- **stream** – **[in]** HIP stream to execute collective on

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **mscclUnloadAlgo**(*mscclAlgoHandle_t* mscclAlgoHandle)

MSCCL Unload Algorithm.

Unload MSCCL algorithm previous loaded using its handle. This API is expected to be called by MSCCL scheduler instead of end users.

Parameters
mscclAlgoHandle – **[in]** Handle to MSCCL algorithm to unload

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclGroupStart**()

Group Start.

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

Returns
Result code. See *Result Codes* for more details.

ncclResult_t **ncclGroupEnd**()

Group End.

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

Returns
Result code. See *Result Codes* for more details.

group **rccl_result_code**

The various result codes that RCCL API calls may return

Enums

enum **ncclResult_t**

Result type.

Return codes aside from ncclSuccess indicate that a call has failed

Values:

enumerator **ncclSuccess**

No error

enumerator **ncclUnhandledCudaError**

Unhandled HIP error

enumerator **ncclSystemError**

Unhandled system error

enumerator **ncclInternalError**

Internal Error - Please report to RCCL developers

enumerator **ncclInvalidArgument**

Invalid argument

enumerator **ncclInvalidUsage**

Invalid usage

enumerator **ncclRemoteError**

Remote process exited or there was a network error

enumerator **ncclInProgress**

RCCL operation in progress

enumerator **ncclNumResults**

Number of result types

group **rccl_config_type**

Structure that allows for customizing Communicator behavior via ncclCommInitRankConfig

Defines

NCCL_CONFIG_INITIALIZER

group **rccl_api_version**

API call that returns RCCL version

group **rccl_api_communicator**

API calls that operate on communicators. Communicators objects are used to launch collective communication operations. Unique ranks between 0 and N-1 must be assigned to each HIP device participating in the same Communicator. Using the same HIP device for multiple ranks of the same Communicator is not supported at this time.

group **rccl_api_errcheck**

API calls that check for errors

group **rccl_api_comminfo**

API calls that query communicator information

group **rccl_api_enumerations**

Enumerations used by collective communication calls

Enums

enum **ncclRedOp_dummy_t**

Dummy reduction enumeration.

Dummy reduction enumeration used to determine value for ncclMaxRedOp

Values:

enumerator **ncclNumOps_dummy**

enum **ncclRedOp_t**

Reduction operation selector.

Enumeration used to specify the various reduction operations ncclNumOps is the number of built-in ncclRedOp_t values and serves as the least possible value for dynamic ncclRedOp_t values constructed by ncclRedOpCreate functions.

ncclMaxRedOp is the largest valid value for ncclRedOp_t and is defined to be the largest signed value (since compilers are permitted to use signed enums) that won't grow sizeof(ncclRedOp_t) when compared to previous RCCL versions to maintain ABI compatibility.

Values:

enumerator **ncclSum**

Sum

enumerator **ncclProd**

Product

enumerator **ncclMax**

Max

enumerator **ncclMin**

Min

enumerator **ncclAvg**

Average

enumerator **ncclNumOps**

Number of built-in reduction ops

enumerator **ncclMaxRedOp**

Largest value for ncclRedOp_t

enum **ncclDataType_t**

Data types.

Enumeration of the various supported datatype

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 **ncclFp8E4M3**

enumerator **ncclFp8E5M2**

enumerator **ncclNumTypes**

group **rccl_api_custom_redop**

API calls relating to creation/destroying custom reduction operator that pre-multiplies local source arrays prior to reduction

Enums

enum **ncclScalarResidence_t**

Location and dereferencing logic for scalar arguments.

Enumeration specifying memory location of the scalar argument. Based on where the value is stored, the argument will be dereferenced either while the collective is running (if in device memory), or before the ncclRedOpCreate() function returns (if in host memory).

Values:

enumerator **ncclScalarDevice**

Scalar is in device-visible memory

enumerator **ncclScalarHostImmediate**

Scalar is in host-visible memory

group **rccl_collective_api**

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

They return when operations have been enqueued on the HIP stream. 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).

group **msccl_api**

API calls relating to the optional MSCCL algorithm datapath

Typedefs

typedef int **mscclAlgoHandle_t**

Opaque handle to MSCCL algorithm.

group **rccl_group_api**

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

Grouping RCCL 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, but not together.

Group semantics also allow to fuse multiple operations on the same device to improve performance (for aggregated collective calls), or to permit concurrent progress of multiple send/receive operations.

dir **src**

page **index**

2.1 Introduction

RCCL (pronounced "Rickle") is a stand-alone library of standard collective communication routines for GPUs, implementing all-reduce, all-gather, reduce, broadcast, reduce-scatter, gather, scatter, and all-to-all. There is also initial support for direct GPU-to-GPU send and receive operations. It has been optimized to achieve high bandwidth on platforms using PCIe, xGMI as well as networking using InfiniBand Verbs or TCP/IP sockets. RCCL supports an arbitrary number of GPUs installed in a single node or multiple nodes, and can be used in either single- or multi-process (e.g., MPI) applications.

The collective operations are implemented using ring and tree algorithms and have been optimized for throughput and latency. For best performance, small operations can be either batched into larger operations or aggregated through the API.

2.2 RCCL API Contents

2.3 RCCL API File

- *nccl.h.in*

CHAPTER

THREE

LICENSE

Attributions

Contains contributions from NVIDIA.

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.

See:

https://github.com/NVIDIA/NVTX

for more information and license details.

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

INDEX

P

R