

---

# **RCCL Documentation**

*Release 0.8*

**Advanced Micro Devices**

**Jun 27, 2022**



# CONTENTS:

- 1 RCCL** **1**
- 1.1 Introduction . . . . . 1
- 2 API** **3**
- 2.1 Communicator Functions . . . . . 3
- 2.2 Collective Communication Operations . . . . . 4
- 2.3 Group Semantics . . . . . 6
- 2.4 Library Functions . . . . . 6
- 2.5 Types . . . . . 6
- 2.6 Enumerations . . . . . 7
- 3 All API** **9**
- 4 Attributions** **17**
- 5 Indices and tables** **19**
- Index** **21**



## **1.1 Introduction**

The RCCL is an AMD port of NCCL.



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 synchronizes 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 implicitly 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 nrank\*recvcount, which means that sendbuff should have a size of at least nrank\*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 `n ranks*sendcount`, which means that `recvbuff` should have a size of at least `n ranks*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)
```

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 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 `n ranks*sendcount`, which means that `recvbuff` should have a size of at least `n ranks*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 `n ranks*recvcount`, which means that `sendbuff` should have a size of at least `n ranks*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 `n ranks*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**

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

RCCL\_MULTIRANKPERGPU

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 implicitly synchronizes 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* **ncclCommInitRankMulti**(*ncclComm\_t* \*comm, int nranks, *ncclUniqueId* commId, int rank, int virtualId)

Creates a new communicator (multi thread/process version) allowing multiple ranks per device.

rank must be between 0 and nranks-1 and unique within a communicator clique. Each rank is associated to a HIP device, which has to be set before calling ncclCommInitRankMulti. Since this version of the function allows multiple ranks to utilize the same HIP device, a unique virtualId per device has to be provided by each calling rank. ncclCommInitRankMulti implicitly synchronizes 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 \***pnccclGetErrorString**(*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* **pnccclRedOpCreatePreMulSum**(*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* **pnccclRedOpDestroy**(*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 implicitly 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 nanks\*recvcount, which means that sendbuff should have a size of at least nanks\*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 nanks\*sendcount, which means that recvbuff should have a size of at least nanks\*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)

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* **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 nanks\*sendcount, which means that recvbuff should have a size of at least nanks\*sendcount elements.

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

*ncclResult\_t* **ncclScatter**(const void \*sendbuff, void \*recvbuff, size\_t recvcnt, *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 n ranks\*recvcnt, which means that sendbuff should have a size of at least n ranks\*recvcnt elements.

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

*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 n ranks\*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 recvcnts[], 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 recvcnts[j] of data from device (j) to be placed at rdispls[j].

sendcounts, sdispls, recvcnts 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.



## ATTRIBUTIONS

Contains contributions from NVIDIA.

Copyright (c) 2015-2020, NVIDIA CORPORATION. All rights reserved.

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>



## INDICES AND TABLES

- genindex
- search





## N

- NCCL\_MAJOR (*C macro*), 9
- NCCL\_MINOR (*C macro*), 9
- NCCL\_PATCH (*C macro*), 9
- NCCL\_SUFFIX (*C macro*), 9
- NCCL\_UNIQUE\_ID\_BYTES (*C macro*), 9
- NCCL\_VERSION (*C macro*), 9
- NCCL\_VERSION\_CODE (*C macro*), 9
- ncclAllGather (*C++ function*), 5, 14
- ncclAllReduce (*C++ function*), 4, 14
- ncclAllToAll (*C++ function*), 5, 15
- ncclAllToAllv (*C++ function*), 15
- ncclBcast (*C++ function*), 4, 13
- ncclBroadcast (*C++ function*), 4, 13
- ncclComm\_t (*C++ type*), 6, 10
- ncclCommAbort (*C++ function*), 3, 13
- ncclCommCount (*C++ function*), 3, 13
- ncclCommCuDevice (*C++ function*), 3, 13
- ncclCommDestroy (*C++ function*), 3, 12
- ncclCommGetAsyncError (*C++ function*), 13
- ncclCommInitAll (*C++ function*), 3, 12
- ncclCommInitRank (*C++ function*), 3, 12
- ncclCommInitRankMulti (*C++ function*), 12
- ncclCommUserRank (*C++ function*), 4, 13
- ncclDataType\_t (*C++ enum*), 7, 11
- ncclDataType\_t::ncclBfloat16 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclChar (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclDouble (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclFloat (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclFloat16 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclFloat32 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclFloat64 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclHalf (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclInt (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclInt32 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclInt64 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclInt8 (*C++ enumerator*), 7, 11
- ncclDataType\_t::ncclNumTypes (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclUInt32 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclUInt64 (*C++ enumerator*), 8, 11
- ncclDataType\_t::ncclUInt8 (*C++ enumerator*), 8, 11
- ncclGather (*C++ function*), 5, 14
- ncclGetErrorString (*C++ function*), 6, 13
- ncclGetUniqueId (*C++ function*), 3, 12
- ncclGetVersion (*C++ function*), 6, 12
- ncclGroupEnd (*C++ function*), 6, 15
- ncclGroupStart (*C++ function*), 6, 15
- ncclRecv (*C++ function*), 5, 14
- ncclRedOp\_dummy\_t (*C++ enum*), 10
- ncclRedOp\_dummy\_t::ncclNumOps\_dummy (*C++ enumerator*), 10
- ncclRedOp\_t (*C++ enum*), 7, 10
- ncclRedOp\_t::ncclAvg (*C++ enumerator*), 7, 10
- ncclRedOp\_t::ncclMax (*C++ enumerator*), 7, 10
- ncclRedOp\_t::ncclMaxRedOp (*C++ enumerator*), 7, 11
- ncclRedOp\_t::ncclMin (*C++ enumerator*), 7, 10
- ncclRedOp\_t::ncclNumOps (*C++ enumerator*), 7, 11
- ncclRedOp\_t::ncclProd (*C++ enumerator*), 7, 10
- ncclRedOp\_t::ncclSum (*C++ enumerator*), 7, 10
- ncclRedOpCreatePreMulSum (*C++ function*), 13
- ncclRedOpDestroy (*C++ function*), 13
- ncclReduce (*C++ function*), 4, 13
- ncclReduceScatter (*C++ function*), 4, 14
- ncclResult\_t (*C++ enum*), 7, 10
- ncclResult\_t::ncclInternalError (*C++ enumerator*), 7, 10
- ncclResult\_t::ncclInvalidArgument (*C++ enumerator*), 7, 10
- ncclResult\_t::ncclInvalidUsage (*C++ enumerator*), 7, 10
- ncclResult\_t::ncclNumResults (*C++ enumerator*),

7, 10  
ncclResult\_t::ncclSuccess (C++ *enumerator*), 7, 10  
ncclResult\_t::ncclSystemError (C++ *enumerator*), 7, 10  
ncclResult\_t::ncclUnhandledCudaError (C++ *enumerator*), 7, 10  
ncclScalarResidence\_t (C++ *enum*), 11  
ncclScalarResidence\_t::ncclScalarDevice (C++ *enumerator*), 12  
ncclScalarResidence\_t::ncclScalarHostImmediate (C++ *enumerator*), 12  
ncclScatter (C++ *function*), 5, 15  
ncclSend (C++ *function*), 5, 14  
ncclUniqueId (C++ *struct*), 6, 9  
ncclUniqueId::internal (C++ *member*), 9

## P

pnccIGetErrorString (C++ *function*), 13  
pnccIRedOpCreatePreMulSum (C++ *function*), 13  
pnccIRedOpDestroy (C++ *function*), 13

## R

RCCL\_ALLTOALLV (C *macro*), 9  
RCCL\_BFLOAT16 (C *macro*), 9  
RCCL\_GATHER\_SCATTER (C *macro*), 9  
RCCL\_MULTIRANKPERGPU (C *macro*), 9