NVIDIA COLLECTIVE COMMUNICATION LIBRARY (NCCL)

PR-08594-001_v | May 2018

API

TABLE OF CONTENTS

Chapter 1. NCCL API

The following sections describe the collective communications methods and operations.

1.1. Communicator Creation And Management Functions

The following functions are public APIs exposed by NVIDIA® Collective Communications Library ™ (NCCL) to create and manage the collective communication operations.

1.1.1. `ncclGetUniqueId`

The `ncclGetUniqueId` function generates an `Id` to be used in the `ncclCommInitRank` function.

The `ncclGetUniqueId` function should be called once. The `Id` should be distributed to all of the ranks in the communicator before calling the `ncclCommInitRank` function.

```
ncclResult_t  ncclGetUniqueId(ncclUniqueId* uniqueId);
```

The following table lists the arguments that are passed to the `ncclGetUniqueId` function.

Type	Argument Name	Description
`ncclUniqueId*`	`uniqueId`	Pointer to an already allocated unique `Id`.

1.1.2. `ncclCommInitRank`

The `ncclCommInitRank` function creates a new communicator object for the current CUDA® device. This function allows for multi-process initialization.

```
ncclResult_t  ncclCommInitRank(ncclComm_t* comm, int nranks, ncclUniqueId
 commId, int
```

```
        rank);
```

The ncclCommInitRank function implicitly synchronizes with other ranks, so it must be called by different threads and processes or use the ncclGroupStart and ncclGroupEnd functions.

The following table lists the arguments that are passed to the ncclCommInitRank function.

Type	Argument Name	Description
ncclComm_t*	comm	Returned communicator.
int	nranks	Number of ranks in the communicator.
ncclUniqueId*	uniqueId	Pointer to a unique Id.
int	rank	The rank associated to the current device. The rank must be between **0** and **nranks-1** and unique within the communicator clique.

1.1.3. ncclCommInitAll

The ncclCommInitAll function creates a full communicator. For example, a clique of communicator objects. The communicator only works within a single process.

```
ncclResult_t  ncclCommInitAll(ncclComm_t* comm, int ndev, const int* devlist);
```

The ncclCommInitAll function returns an array of **ndev** newly initialized communicators in **comm**. The argument name **comm**, should be pre-allocated with the size of at least **ndev*sizeof(ncclComm_t)**. If **devlist** is **NULL**, the first **ndev** CUDA devices are used. The order of **devlist** defines the user order of the devices within the communicator.

The following table lists the arguments that are passed to the ncclCommInitAll function.

Type	Argument Name	Description
ncclComm_t*	comm	Returned array of communicators. The comm argument should be pre-allocated with a size of at least: **ndev*sizeof(ncclComm_t)**.
int	ndev	Number of ranks or devices in the communicator.
const int*	devlist	A list of CUDA devices to associate with each rank.

Type	Argument Name	Description
		Should be an array of **ndev** integers.

1.1.4. ncclCommDestroy

The ncclCommDestroy function frees resources that are allocated to a communicator object.

```
ncclResult_t  ncclCommDestroy(ncclComm_t comm);
```

The following table lists the arguments that are passed to the ncclCommDestroy function.

Type	Argument Name	Description
ncclComm_t	comm	Communicator object to free.

1.1.5. ncclCommCount

The ncclCommCount function returns the number of ranks in a communicator.

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

The following table lists the arguments that are passed to the ncclCommCount function.

Type	Argument Name	Description
ncclComm_t	comm	Communicator object.
int*	count	Number of ranks returned.

1.1.6. ncclCommCuDevice

The ncclCommCuDevice function returns the CUDA device associated with a communicator object.

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

The following table lists the arguments that are passed to the ncclCommCuDevice function.

Type	Argument Name	Description
ncclComm_t	comm	Communicator object.
int*	count	CUDA device returned.

1.1.7. ncclCommUserRank

The ncclCommUserRank function returns the rank of a communicator object.

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

The following table lists the arguments that are passed to the `ncclCommUserRank` function.

Type	Argument Name	Description
`ncclComm_t`	`comm`	Communicator object.
`int*`	`rank`	Rank returned.

1.2. Collective Communication Functions

The following NCCL APIs provide some commonly used collective operations.

1.2.1. `ncclAllReduce`

The `ncclAllReduce` function reduces data arrays of length count in **`sendbuff`** using **`op`** operation and leaves identical copies of the result on each **`recvbuff`**.

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

The following table lists the arguments that are passed to the `ncclAllReduce` function.

Type	Argument Name	Description
`const void*`	`sendbuff`	Pointer to the data to read from.
`void*`	`recvbuff`	Pointer to the data to write to.
`size_t`	`count`	Number of elements to process.
`ncclDataType_t`	`datatype`	Type of element.
`ncclRedOp_t`	`op`	Operation to perform on each element.
`ncclComm_t`	`comm`	Communicator object.
`cudaStream_t`	`stream`	CUDA stream to run the operation on.

1.2.2. `ncclBroadcast`

The `ncclBroadcast` function copies the count values from the root rank to all ranks.

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

The **`ncclBcast`** function is a legacy in-place version of **`ncclBroadcast`** in a similar fashion to **`MPI_Bcast`**. A call to **`ncclBcast`** (**`buff`**, **`count`**, **`datatype`**, **`root`**, **`comm`**, **`stream`**) is equivalent to **`ncclBroadcast`** (**`buff`**, **`count`**, **`datatype`**, **`root`**, **`comm`**, **`stream`**).

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

The following table lists the arguments that are passed to the `ncclBroadcast` function.

Type	Argument Name	Description
`const void*`	`sendbuff`	Pointer to the data to read from.
`void*`	`recvbuff`	Pointer to the data to read to.
`size_t`	`count`	Number of elements to process.
`ncclDataType_t`	`datatype`	Type of element.
`int`	`root`	Rank of the root of the operation.
`ncclComm_t`	`comm`	Communicator object.
`cudaStream_t`	`stream`	CUDA stream to run the operation on.

1.2.3. `ncclReduce`

The `ncclReduce` function reduces data arrays of length count in **`sendbuff`** into **`recvbuff`** using the **`op`** operation.

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

The following table lists the arguments that are passed to the `ncclReduce` function.

Type	Argument Name	Description
`const void*`	`sendbuff`	Pointer to the data to read from.
`void*`	`recvbuff`	Pointer to the data to write to.
`size_t`	`count`	Number of elements to process.
`ncclDataType_t`	`datatype`	Type of element.

Type	Argument Name	Description
ncclRedOp_t	op	Operation to perform on each element.
int	root	Rank of the root of the operation.
ncclComm_t	comm	Communicator object.
cudaStream_t	stream	CUDA stream to run the operation on.

1.2.4. ncclAllGather

The ncclAllGather function gathers **sendcount** values from other GPUs into **recvbuff**, receiving data from rank **i** at offset **i*sendcount**.

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

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

The following table lists the arguments that are passed to the ncclAllGather function.

Type	Argument Name	Description
const void*	sendbuff	Pointer to the data to read from.
void*	recvbuff	Pointer to the data to write to. This should be the size of **sendcount*nranks**.
size_t	sendcount	Number of elements sent per rank.
ncclDataType_t	datatype	Type of element.
int	root	Rank of the root of the operation.
ncclComm_t	comm	Communicator object.
cudaStream_t	stream	CUDA stream to run the operation on.

1.2.5. ncclReduceScatter

The `ncclReduceScatter` function reduces data in **sendbuff** using the **op** operation and leaves the reduced result scattered over the devices so that the **recvbuff** on rank **i** will contain the **i-th** block of the result.

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

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

The following table lists the arguments that are passed to the `ncclReduceScatter` function.

Type	Argument Name	Description
`const void*`	`sendbuff`	Pointer to the data to read from. This should be the size of **recvcount*nranks**.
`void*`	`recvbuff`	Pointer to the data to write to.
`size_t`	`recvcount`	Number of elements to receive by each rank.
`ncclDataType_t`	`datatype`	Type of element.
`ncclRedOp_t`	`op`	Operation to perform on each element.
`ncclComm_t`	`comm`	Communicator object.
`cudaStream_t`	`stream`	CUDA stream to run the operation on.

1.3. Group Calls

Group primitives define the behavior of the current thread to avoid blocking. They can therefore be used from multiple threads independently.

1.3.1. ncclGroupStart

The `ncclGroupStart` call starts a group call.

All subsequent calls to NCCL may not block due to inter-CPU synchronization.

```
ncclResult_t ncclGroupStart();
```

1.3.2. ncclGroupEnd

The `ncclGroupEnd` call ends a group call.

The ncclGroupEnd call returns when all operations since ncclGroupStart have been processed. This means communication primitives have been enqueued to the provided streams, but are not necessary complete. When used with ncclCommInitRank, it means all communicators have been initialized and are ready to be used.

When the ncclGroupEnd call is used with the **ncclCommInitRank** function, the ncclGroupEnd call waits for all communicators to be initialized.

```
ncclResult_t ncclGroupEnd();
```

1.4. Types

The following types are used by the CUDA library. These types are useful when configuring your collective operations.

1.4.1. ncclDataType_t

NCCL defines the following integral and floating data-types.

Data-Type	Description
ncclInt8, ncclChar	Signed 8-bits integer.
ncclUint8	Unsigned 8-bits integer.
ncclInt32, ncclInt	Signed 32-bits integer.
ncclUint32	Unsigned 32-bits integer.
ncclInt64	Signed 64-bits integer.
ncclUint64	Unsigned 64-bits integer.
ncclFloat16, ncclHalf	16-bits floating point number (half precision)
ncclFloat32, ncclFloat	32-bits floating point number (single precision)
ncclFloat64, ncclDouble	64-bits floating point number (double precision)

1.4.2. ncclRedOp_t

NCCL defines the following reduction operations.

Reduction Operation	Description
ncclSum	Perform a sum (+) operation.
ncclProd	Perform a product (*) operation.
ncclMin	Perform a min operation.

Reduction Operation	Description
ncclMax	Perform a max operation.

1.4.3. ncclResult_t

NCCL functions always return an error code of type ncclResult_t.

If the *NCCL_DEBUG* environment variable is set to **WARN**, whenever a function returns an error, NCCL should print the reason.

Return Code	Description
ncclSuccess	The operations completed successfully.
ncclUnhandledCudaError	A call to CUDA returned a fatal error for the NCCL operation.
ncclSystemError	A call to the system returned a fatal error for the NCCL operation.
ncclInternalError	NCCL experienced an internal error.
ncclInvalidArgument	The user has supplied an invalid argument.
ncclInvalidUsage	The user has used NCCL in an invalid manner.

1.5. Constants

NCCL defines two constants **NCCL_MAJOR** and **NCCL_MINOR** to help distinguish between API changes, in particular between NCCL 1.x and NCCL 2.x.

Notice

Trademarks

Copyright

www.nvidia.com