NVIDIA COLLECTIVE COMMUNICATION LIBRARY (NCCL)

DU-08730-210_v01 | January 2019

Installation Guide

TABLE OF CONTENTS

Chapter 1. Overview 1
Chapter 2. Prerequisites 3
2.1. Software Requirements 3
2.2. Hardware Requirements 3
Chapter 3. Installing NCCL 4
3.1. Ubuntu 14.04 LTS And Ubuntu 16.04 LTS 4
3.2. RHEL/CentOS 7 5
3.3. Other Distributions 6
Chapter 4. Using NCCL 7
Chapter 5. Migrating From NCCL 1 To NCCL 2 8
Chapter 6. Troubleshooting 10
6.1. Support 10

Chapter 1. OVERVIEW

The NVIDIA® Collective Communications Library ™ (NCCL) (pronounced "Nickel") is a library of multi-GPU collective communication primitives that are topology-aware and can be easily integrated into applications.

Collective communication algorithms employ many processors working in concert to aggregate data. NCCL is not a full-blown parallel programming framework; rather, it is a library focused on accelerating collective communication primitives. The following collective operations are currently supported:

- **`AllReduce`**
- **`Broadcast`**
- **`Reduce`**
- **`AllGather`**
- **`ReduceScatter`**

Tight synchronization between communicating processors is a key aspect of collective communication. CUDA® based collectives would traditionally be realized through a combination of CUDA memory copy operations and CUDA kernels for local reductions. NCCL, on the other hand, implements each collective in a single kernel handling both communication and computation operations. This allows for fast synchronization and minimizes the resources needed to reach peak bandwidth.

NCCL conveniently removes the need for developers to optimize their applications for specific machines. NCCL provides fast collectives over multiple GPUs both within and across nodes. It supports a variety of interconnect technologies including PCIe, NVLink™ , InfiniBand Verbs, and IP sockets. NCCL also automatically patterns its communication strategy to match the system's underlying GPU interconnect topology.

Next to performance, ease of programming was the primary consideration in the design of NCCL. NCCL uses a simple C API, which can be easily accessed from a variety of programming languages. NCCL closely follows the popular collectives API defined by MPI (Message Passing Interface). Anyone familiar with MPI will thus find NCCL API very natural to use. In a minor departure from MPI, NCCL collectives take a "stream" argument which provides direct integration with the CUDA programming model. Finally, NCCL is compatible with virtually any multi-GPU parallelization model, for example:

- single-threaded
- multi-threaded, for example, using one thread per GPU
- multi-process, for example, MPI combined with multi-threaded operation on GPUs

NCCL has found great application in deep learning frameworks, where the **`AllReduce`** collective is heavily used for neural network training. Efficient scaling of neural network training is possible with the multi-GPU and multi node communication provided by NCCL.

Chapter 2. PREREQUISITES

2.1. Software Requirements

Ensure your environment meets the following software requirements:

- glibc 2.19 or higher
- CUDA 8.0 or higher

2.2. Hardware Requirements

NCCL supports all CUDA devices with a compute capability of 3.0 and higher. For the compute capability of all NVIDIA GPUs, check: CUDA GPUs.

Chapter 3. INSTALLING NCCL

In order to download NCCL, ensure you are registered for the NVIDIA Developer Program.

1. Go to: NVIDIA NCCL home page.
2. Click **Download**.
3. Complete the short survey and click **Submit**.
4. Accept the Terms and Conditions. A list of available download versions of NCCL displays.
5. Select the NCCL version you want to install. A list of available resources displays. Refer to the following sections to choose the correct package depending on the Linux distribution you are using.

3.1. Ubuntu 14.04 LTS And Ubuntu 16.04 LTS

Installing NCCL on Ubuntu requires you to first add a repository to the APT system containing the NCCL packages, then installing the NCCL packages through APT. There are two repositories available; a local repository and a network repository. Choosing the later is recommended to easily retrieve upgrades when newer versions are posted.

1. Install the repository.
   - For the local NCCL repository:

     ```
     sudo dpkg -i nccl-repo-<version>.deb
     ```
   - For the network repository:

     ```
     sudo dpkg -i nvidia-machine-learning-repo-<version>.deb
     ```
2. Update the APT database:

   ```
   sudo apt update
   ```

3. Install the **libnccl2** package with APT. Additionally, if you need to compile applications with NCCL, you can install the **libnccl-dev** package as well:

If you are using the network repository, the following command will upgrade CUDA to the latest version.

```
sudo apt install libnccl2 libnccl-dev
```

If you prefer to keep an older version of CUDA, specify a specific version, for example:

```
sudo apt install libnccl2=2.0.0-1+cuda8.0 libnccl-dev=2.0.0-1+cuda8.0
```

Refer to the download page for exact package versions.

3.2. RHEL/CentOS 7

Installing NCCL on RHEL 7 or CentOS 7 requires you to first add a repository to the YUM system containing the NCCL packages, then installing the NCCL packages through YUM. There are two repositories available; a local repository and a network repository. Choosing the later is recommended to easily retrieve upgrades when newer versions are posted.

1. Install the repository.
   - For the local NCCL repository:

     ```
     sudo rpm -i nccl-repo-<version>.rpm
     ```
   - For the network repository:

     ```
     sudo rpm -i nvidia-machine-learning-repo-<version>.rpm
     ```
2. Update the YUM database:

```
sudo yum update
```

3. Install the **libnccl2** package with YUM. Additionally, if you need to compile applications with NCCL, you can install the **libnccl-devel** package and optionally the **libnccl-static** package if you intend to link NCCL statically in your application:

If you are using the network repository, the following command will upgrade CUDA to the latest version.

```
sudo yum install libnccl libnccl-devel libnccl-static
```

If you prefer to keep an older version of CUDA, specify a specific version, for example:

```
sudo yum install libnccl-2.4.2-1+cuda10.0
libnccl-devel-2.4.2-1+cuda10.0 libnccl-static-2.4.2-1+cuda10.0
```

Refer to the download page for exact package versions.

3.3. Other Distributions

Download the tar file package. For more information see Installing NCCL.

1. Extract the NCCL package to your home directory or in **/usr/local** if installed as root for all users:

```
# cd /usr/local
# tar xvf nccl-<version>.txz
```

2. When compiling applications, specify the directory path to where you installed NCCL, for example **/usr/local/nccl-<version>/**.

Chapter 4. USING NCCL

Using NCCL is similar to using any other library in your code. For example:

1. Install the NCCL library onto your system.
   For more information, see Downloading NCCL.
2. Modify your application to link to that library.
3. Include the header file `nccl.h` in your application.
4. Create a communicator.
   For more information, see *Creating a Communicator* in the NCCL Developer Guide.
5. Familiarize yourself with the NCCL API documentation to maximize your usage performance.

Chapter 5. MIGRATING FROM NCCL 1 TO NCCL 2

If you are using NCCL 1.x and want to move to NCCL 2.x, be aware that the APIs have changed slightly. NCCL 2.x supports all of the collectives that NCCL 1.x supports, but with slight modifications to the API.

In addition, NCCL 2.x also requires the usage of the Group API when a single thread manages NCCL calls for multiple GPUs.

The following list summarizes the changes that may be required in usage of NCCL API when using an application has a single thread that manages NCCL calls for multiple GPUs, and is ported from NCCL 1.x to 2.x:

Initialization

In 1.x, NCCL had to be initialized using **`ncclCommInitAll`** at a single thread or having one thread per GPU concurrently call **`ncclCommInitRank`**. NCCL 2.x retains these two modes of initialization. It adds a new mode with the Group API where **`ncclCommInitRank`** can be called in a loop, like a communication call, as shown below. The loop has to be guarded by the Group start and stop API.

```
ncclGroupStart();
 for (int i=0; i<ngpus; i++) {
   cudaSetDevice(i);
   ncclCommInitRank(comms+i, ngpus, id, i);
 }
 ncclGroupEnd();
```

Communication

In NCCL 2.x, the collective operation can be initiated for different devices by making calls in a loop, on a single thread. This is similar to the usage in NCCL 1.x. However, this loop has to be guarded by the Group API in 2.x. Unlike in 1.x, the application does not have to select the relevant CUDA device before making the communication API call. NCCL runtime internally selects the device associated with the NCCL communicator handle. For example:

```
ncclGroupStart();
 for (int i=0; i<nLocalDevs; i++) {
```

```
  ncclAllReduce(..., comm[i], stream[i];
}
ncclGroupEnd();
```

When using only one device per thread or one device per process, the general usage of API remains unchanged from NCCL 1.x to 2.x. Group API is not required in this case.

Counts

Counts provided as arguments are now of type **`size_t`** instead of **`integer`**.

In-place usage for `AllGather` and `ReduceScatter`

For more information, see *In-Place Operations* in the NCCL Developer Guide.

`AllGather` arguments order

The **`AllGather`** function had its arguments reordered. The prototype changed from:

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

to:

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

The **`recvbuff`** argument has been moved after the **`sendbuff`** argument to be consistent with all the other operations.

Datatypes

New datatypes have been added in NCCL 2.x. The ones present in NCCL 1.x did not change and are still usable in NCCL 2.x.

Error codes

Error codes have been merged into the **`ncclInvalidArgument`** category and have been simplified. A new **`ncclInvalidUsage`** code has been created to cover new programming errors.

Chapter 6. TROUBLESHOOTING

6.1. Support

Register for the NVIDIA Developer Program to report bugs, issues and make requests for feature enhancements. For more information, see: https://developer.nvidia.com/developer-program.

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, cuFFT, cuSPARSE, DALI, DIGITS, DGX, DGX-1, Jetson, Kepler, NVIDIA Maxwell, NCCL, NVLink, Pascal, Tegra, TensorRT, and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2019 NVIDIA Corporation. All rights reserved.