NVIDIA COLLECTIVE COMMUNICATION LIBRARY (NCCL)

RN-08645-000_v01 | June 2018

Release Notes

TABLE OF CONTENTS

Chapter 1. NCCL 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. NCCL RELEASE 2.2.13

Key Features and Enhancements

There were no new features and enhancements in this release.

Using NCCL 2.2.13

Ensure you are familiar with the following notes when using this release.

- If NCCL returns an error code, set the environment variable **NCCL_DEBUG** to **WARN** to receive an explicit error message.
- The NCCL 2.x API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.
- Starting in 2.2, NCCL supports collective aggregation. You can put multiple NCCL collective operations in between **ncclGroupStart()** and **ncclGroupEnd()** to enable this feature.

Known Issues

- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause **ncclCommInitRank** to fail while establishing IPCs (**cudaIpcOpenMemHandle**). This problem does not appear when using only processes or only threads. This issue is fixed in recent driver versions, therefore, consider updating to the latest driver.
- NCCL uses CUDA 9 cooperative group launch by default, which may induce increased latencies in multi-threaded programs. See the **NCCL_LAUNCH_MODE** knob to restore the original behavior.
- Driver version 390 can cause data corruption when used together with GPU Direct RDMA. Disabling GPU Direct RDMA by setting **NCCL_IB_CUDA_SUPPORT=0**, or upgrading to 396.26 or newer driver should resolve the issue.

Fixed Issues

- Fix crash in child processes after calling **`ncclCommDestroy`**.

Chapter 3. NCCL RELEASE 2.2.12

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added support for collective operations aggregation.
- Added **`ncclBroadcast`** function.

Using NCCL 2.2.12

Ensure you are familiar with the following notes when using this release.

- If NCCL returns an error code, set the environment variable **`NCCL_DEBUG`** to **`WARN`** to receive an explicit error message.
- The NCCL 2.x API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.
- Starting in 2.2, NCCL supports collective aggregation. You can put multiple NCCL collective operations in between **`ncclGroupStart()`** and **`ncclGroupEnd()`** to enable this feature.

Known Issues

- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause **`ncclCommInitRank`** to fail while establishing IPCs (**`cudaIpcOpenMemHandle`**). This problem does not appear when using only processes or only threads. This issue is fixed in recent driver versions, therefore, consider updating to the latest driver.
- NCCL uses CUDA 9 cooperative group launch by default, which may induce increased latencies in multi-threaded programs. See the **`NCCL_LAUNCH_MODE`** knob to restore the original behavior.

- Driver version 390 and later can cause data corruption when used together with GPU Direct RDMA. Disabling GPU Direct RDMA by setting **`NCCL_IB_CUDA_SUPPORT=0`** or reverting to driver 387 should resolve the issue.

Fixed Issues

- No longer clear the CPU affinity during initialization functions
- Fix various large scale issues
- Reduce the size of the library
- Fix crash or hang with PyTorch relative to the usage of calls to fork

Chapter 4. NCCL RELEASE 2.1.15

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added a variable to control InfiniBand Traffic Class and Retry Count.

Using NCCL 2.1.15

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.x API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.
- Starting in 2.2, NCCL supports collective aggregation. You can put multiple collectives in between **`ncclGroupStart()`** and **`ncclGroupEnd()`** to enable this feature.

Known Issues

- If NCCL returns an error code, set the environment variable **`NCCL_DEBUG`** to **`WARN`** to receive an explicit error message.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause **`ncclCommInitRank`** to fail while establishing IPCs (**`cudaIpcOpenMemHandle`**). This problem does not appear when using only processes or only threads.
- NCCL uses CUDA 9 cooperative group launch by default, which may induce increased latencies in multi-threaded programs. See the **`NCCL_LAUNCH_MODE`** knob to restore the original behavior.

Fixed Issues

- Fixed CPU usage and scheduling of NCCL network threads.

- Fixed CUDA launch crash when mixing different types of GPUs in a node.
- Fixed a performance problem on Skylake CPUs.
- Fixed hanging issues with cudaFree and inter-node communication.
- Restored library installation path to **`/usr/lib/x86_64-linux-gnu`** in debian packages.
- Fixed RoCEv2 failure when using a non-zero GID.
- No longer link to **`stdc++`** library statically as this can cause issues with C++ applications.
- Fixed PyTorch hanging issues when using multiple rings and many back-to-back broadcast operations.

Chapter 5. NCCL RELEASE 2.1.4

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added support for InfiniBand GID selection, enabling the use of RoCE v2.
- Added support for InfiniBand Service Level (SL) selection.

Using NCCL 2.1.4

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.x API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.

Known Issues

- If NCCL returns an error code, set the environment variable **`NCCL_DEBUG to WARN`** to receive an explicit error message.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause **`ncclCommInitRank`** to fail while establishing IPCs (**`cudaIpcOpenMemHandle`**). This problem does not appear when using only processes or only threads.
- NCCL uses CUDA® 9 cooperative group launch by default, which may induce increased latencies in multi-threaded programs. See the **`NCCL_LAUNCH_MODE`** knob in the NCCL Developer Guide to restore the original behavior.
- NCCL 2.1.4-1 embeds **`libstdc++`** and exports its symbols. This can break C++ applications.

Fixed Issues

- Fixed bug causing CUDA IPC to fail in some situations.

- Fixed bug causing a crash when p2p mappings are exhausted instead of returning an error.

Chapter 6. NCCL RELEASE 2.1.2

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- New algorithms for improved latency communication
- RoCE support

Using NCCL 2.1.2

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.x API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.

Known Issues

- If NCCL returns an error code, set the environment variable **`NCCL_DEBUG to WARN`** to receive an explicit error message.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause **`ncclCommInitRank`** to fail while establishing IPCs (**`cudaIpcOpenMemHandle`**). This problem does not appear when using only processes or only threads.
- NCCL uses CUDA 9 cooperative group launch by default, which may induce increased latencies in multi-threaded programs. See the **`NCCL_LAUNCH_MODE`** knob in the NCCL Developer Guide to restore the original behavior.

Fixed Issues

- NCCL now uses CUDA 9 cooperative groups to launch the CUDA kernels, fixing a long-standing issue with CUDA and NCCL operations being interleaved and potentially causing hangs.

Chapter 7. NCCL RELEASE 2.0.5

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- NCCL 2.0.5 provides support for intra-node and inter-node communication.
- NCCL optimizes intra-node communication using NVLink, PCI express, and shared memory.
- Between nodes, NCCL implements fast transfers over sockets or InfiniBand verbs.
- GPU-to-GPU and GPU-to-Network direct transfers, using the GPU Direct technology, are extensively used when the hardware topology permits it.

Using NCCL 2.0.5

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.0 API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.
- NCCL 2.0.5 has the new configuration file support. The NCCL environment variables can now be set in **`~/.nccl.conf`** and **`/etc/nccl.conf`**.
- Values defined in **`~/.nccl.conf`** take precedence over ones in **`/etc/nccl.conf`**.
- The syntax for each line of the NCCL configuration file is **`<NCCL_VAR_NAME>=<VALUE>`**.

Known Issues

- If NCCL returns any error code, set the environment variable `NCCL_DEBUG` to **`WARN`** to receive an explicit error message.
- RoCE support is experimental.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause `ncclCommInitRank` to fail while

establishing IPCs (`cudaIpcOpenMemHandle`). This problem does not appear when using only processes or only threads.

Chapter 8. NCCL RELEASE 2.0.4

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- NCCL 2.0.4 provides support for intra-node and inter-node communication.
- NCCL optimizes intra-node communication using NVLink, PCI express, and shared memory.
- Between nodes, NCCL implements fast transfers over sockets or InfiniBand verbs.
- GPU-to-GPU and GPU-to-Network direct transfers, using the GPU Direct technology, are extensively used when the hardware topology permits it.

Using NCCL 2.0.4

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.0 API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.
- NCCL 2.0.4 has the new configuration file support. The NCCL environment variables can now be set in **`~/.nccl.conf`** and **`/etc/nccl.conf`**.
- Values defined in **`~/.nccl.conf`** take precedence over ones in **`/etc/nccl.conf`**.
- The syntax for each line of the NCCL configuration file is **`<NCCL_VAR_NAME>=<VALUE>`**.

Known Issues

- If NCCL returns any error code, set the environment variable `NCCL_DEBUG` to **`WARN`** to receive an explicit error message.
- RoCE is not supported.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause `ncclCommInitRank` to fail while

establishing IPCs (`cudaIpcOpenMemHandle`). This problem does not appear when using only processes or only threads.

Chapter 9. NCCL RELEASE 2.0.2

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- NCCL 2.0.2 provides support for intra-node and inter-node communication.
- NCCL optimizes intra-node communication using NVLink, PCI express, and shared memory.
- Between nodes, NCCL implements fast transfers over sockets or InfiniBand verbs.
- GPU-to-GPU and GPU-to-Network direct transfers, using the GPU Direct technology, is extensively used when the hardware topology permits it.

Using NCCL 2.0.2

Ensure you are familiar with the following notes when using this release.

- The NCCL 2.0 API is different from NCCL 1.x. Some porting may be needed for NCCL 1.x applications to work correctly. Refer to the migration documentation in the NCCL Developer Guide.

Known Issues

- NCCL 2.0.2 is known to not work with CUDA driver 384.40 and later.
- If NCCL returns any error code, set the environment variable `NCCL_DEBUG` to **`WARN`** to receive an explicit error message.
- NCCL 2.0.2 does not support RoCE, that is, InfiniBand cards using Ethernet as link layer. The presence of an RoCE card on a node will make NCCL fail even when run within the node.
- Using multiple processes in conjunction with multiple threads to manage the different GPUs may in some cases cause `ncclCommInitRank` to fail while establishing IPCs (`cudaIpcOpenMemHandle`). This problem does not appear when using only processes or only threads.

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, 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