NVIDIA Collective Communication Library (NCCL)

Release Notes

RN-08645-000_v2.7.6 | January 2021

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 2.8.3

This is the NCCL 2.8.3 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Compatibility

NCCL 2.8.3 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 10.1, CUDA 10.2, CUDA 11.0, CUDA 11.1, and CUDA 11.2.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Optimized Tree performance on A100
- Improved performance for aggregated operations
- Improved performance for all-to-all operations at scale
- Reduced memory usage for all-to-all operations at scale
- Optimized all-to-all performance on DGX-1

Known Issues

Send/receive operations have a number of limitations:

- Using send/receive operations in combination to launch work on multiple GPUs from a single process can fail or hang if the GPUs process different amounts of data. Setting `NCCL_LAUNCH_MODE=PARALLEL` can work around the issue, but can also cause other problems. For more information, see the NCCL User Guide section **Troubleshooting** > **Known Issues** > Concurrency Between NCCL and CUDA calls.

Fixed Issues

The following issues have been resolved in NCCL 2.8.3:

- Hang in LL128 protocol after 2^31 steps.

- Topology injection error when using fewer GPUs than described. (github issue #379)
- Protocol mismatch causing hangs or crashes when using one GPU per node. (github issue #394)

Chapter 3. NCCL Release 2.7.8

This is the NCCL 2.7.8 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Compatibility

NCCL 2.7.8 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 10.1, CUDA 10.2, CUDA 11.0, and CUDA 11.1.

Known Issues

Send/receive operations have a number of limitations:

- Using send/receive operations in combination to launch work on multiple GPUs from a single process can fail or hang if the GPUs process different amounts of data. Setting `NCCL_LAUNCH_MODE=PARALLEL` can work around the issue, but can also cause other problems. For more information, see the NCCL User Guide section **Troubleshooting** > **Known Issues** > Concurrency Between NCCL and CUDA calls.
- Aggregation is not supported on a given source-destination pair, meaning that there can only be one send/receive per source-destination pair.
- Each source-destination pair allocates a dedicated fifo of 4 MB (see `NCCL_BUFFSIZE` variable) which can use a significant amount of GPU memory at scale.
- When using GPU Direct RDMA, each point-to-point connection will also use resources on the GPU PCI address space, which we might run out of on some models. If that happens, consider disabling GPU Direct RDMA (`NCCL_NET_GDR_LEVEL=0`) or reduce the per-peer buffer size (`NCCL_BUFFSIZE`).
- Send/receive operations are not yet optimized for the DGX-1 NVLink topology.

Fixed Issues

The following issues have been resolved in NCCL 2.7.8:

- Resolved "Collective Mismatch" errors reported erroneously when using send/recv operations.

Chapter 4. NCCL Release 2.7.6

This is the NCCL 2.7.6 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Compatibility

NCCL 2.7.6 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 10.1, CUDA 10.2, and CUDA 11.0.

Known Issues

Send/receive operations have a number of limitations:

- Using send/receive operations in combination to launch work on multiple GPUs from a single process can fail or hang if the GPUs process different amounts of data. Setting `NCCL_LAUNCH_MODE=PARALLEL` can work around the issue, but can also cause other problems. For more information, see the NCCL User Guide section **Troubleshooting** > **Known Issues** > Concurrency Between NCCL and CUDA calls.
- Aggregation is not supported on a given source-destination pair, meaning that there can only be one send/receive per source-destination pair.
- Each source-destination pair allocates a dedicated fifo of 4 MB (see `NCCL_BUFFSIZE` variable) which can use a significant amount of GPU memory at scale.
- When using GPU Direct RDMA, each point-to-point connection will also use resources on the GPU PCI address space, which we might run out of on some models. If that happens, consider disabling GPU Direct RDMA (`NCCL_NET_GDR_LEVEL=0`) or reduce the per-peer buffer size (`NCCL_BUFFSIZE`).
- Send/receive operations are not yet optimized for the DGX-1 NVLink topology.

Fixed Issues

The following issues have been resolved in NCCL 2.7.6:

- Fixed crash when NVswitch is not visible inside a virtual machine.

Chapter 5. NCCL Release 2.7.5

This is the NCCL 2.7.5 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Compatibility

NCCL 2.7.5 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 10.1, CUDA 10.2, and CUDA 11.0.

Known Issues

Send/receive operations have a number of limitations:

- Using send/receive operations in combination to launch work on multiple GPUs from a single process can fail or hang if the GPUs process different amounts of data. Setting `NCCL_LAUNCH_MODE=PARALLEL` can work around the issue, but can also cause other problems. For more information, see the NCCL User Guide section **Troubleshooting** > **Known Issues** > Concurrency Between NCCL and CUDA calls.
- Aggregation is not supported on a given source-destination pair, meaning that there can only be one send/receive per source-destination pair.
- Each source-destination pair allocates a dedicated fifo of 4 MB (see `NCCL_BUFFSIZE` variable) which can use a significant amount of GPU memory at scale.
- When using GPU Direct RDMA, each point-to-point connection will also use resources on the GPU PCI address space, which we might run out of on some models. If that happens, consider disabling GPU Direct RDMA (`NCCL_NET_GDR_LEVEL=0`) or reduce the per-peer buffer size (`NCCL_BUFFSIZE`).
- Send/receive operations are not yet optimized for the DGX-1 NVLink topology.
- Running inside a virtual machine on an NVswitch platform can cause a crash if NVswitch is not visible inside the virtual machine.

Fixed Issues

The following issues have been resolved in NCCL 2.7.5:

- Minor fixes for A100 platforms.
- Add proper message for invalid `GroupEnd` call.

Chapter 6. NCCL Release 2.7.3

This is the NCCL 2.7.3 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added support for A100 GPU and related platforms
- Added support for CUDA 11
- Added support for send/receive operations (beta)

Compatibility

NCCL 2.7.3 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 10.1, CUDA 10.2, and CUDA 11.0.

Known Issues

Send/receive operations have a number of limitations:

- Using send/receive operations in combination to launch work on multiple GPUs from a single process can fail or hang if the GPUs process different amounts of data. Setting `NCCL_LAUNCH_MODE=PARALLEL` can work around the issue, but can also cause other problems. For more information, see the NCCL User Guide section **Troubleshooting** > **Known Issues** > Concurrency Between NCCL and CUDA calls.
- Aggregation is not supported on a given source-destination pair, meaning that there can only be one send/receive per source-destination pair.
- Each source-destination pair allocates a dedicated fifo of 4 MB (see `NCCL_BUFFSIZE` variable) which can use a significant amount of GPU memory at scale.
- When using GPU Direct RDMA, each point-to-point connection will also use resources on the GPU PCI address space, which we might run out of on some models. If that happens,

consider disabling GPU Direct RDMA (`NCCL_NET_GDR_LEVEL=0`) or reduce the per-peer buffer size (`NCCL_BUFFSIZE`).

- Send/receive operations are not yet optimized for the DGX-1 NVLink topology.

Fixed Issues

The following issues have been resolved in NCCL 2.7.3:

- Fixed crash when only a subset of GPUs are visible within a container (#326).

Chapter 7. NCCL Release 2.6.4

This is the NCCL 2.6.4 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added support for in-network collectives (beta)
- Added support for adaptive routing on Infiniband
- Added support for topology dump/injection through XML files
- Added speed detection for PCI, Infiniband and Ethernet cards
- Added CPU detection for AMD CPUs and ARM

Chapter 8. NCCL Release 2.5.6

This is the NCCL 2.5.6 release notes. For previous NCCL release notes, refer to the NCCL Archives.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Added new protocol to improve performance on small operations.
- Improved topology detection and tree/ring creation (#179, #262).
- Improved multi-node tree performance by sending/receiving from different GPUs.
- Added model-based tuning to switch between the different algorithms and protocols.
- Reworked P2P/SHM detection in containers (#155, #248).
- Added detection for duplicate CUDA devices and return an error (#231).
- Added tuning for Google Cloud's gVNIC platform.

Compatibility

NCCL 2.5.6 has been tested with the following:

- Deep learning framework containers. Refer to the Support Matrix for the supported container version.
- This NCCL release supports CUDA 9.0, CUDA 10.0, CUDA 10.1, and CUDA 10.2.

Fixed Issues

The following issues have been resolved in NCCL 2.5.6:

- Sporadic NCCL error "ring 0 does not loop back to start" (#179).
- NCCL doesn't form proper rings on GCP V100s (#262).

Chapter 9. NCCL Release 2.4.8

This is the NCCL 2.4.8 release notes. This release includes fixes from the previous NCCL 2.4.x releases as well as the following additional changes. For previous NCCL release notes, see the archived NCCL Release Notes.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Improved socket transport performance by splitting transfer over multiple sockets.

> **Note:** This feature adds two new environment variables `NCCL_SOCKET_NTHREADS` and `NCCL_NSOCKS_PERTHREAD` for users to tune NCCL performance on socket-based networks. See the NCCL documentation for more details.

Compatibility

NCCL 2.4.8 has been tested with the following:

- Deep learning framework 19.05 containers
- This NCCL release supports; CUDA 9.0, CUDA 9.2 , CUDA 10.0, and CUDA 10.1.

Fixed Issues

The following issues have been resolved in NCCL 2.4.8:

- Suboptimal performance with TCP over high bandwidth networks. (GitHub issue #209)

Known Issues

- On single node Power systems with 4 GPUs, some performance regressions have been observed compared to NCCL 2.4.2. These will be addressed in future NCCL releases.
- By default, NCCL does not enable direct P2P communication through different PCIe root ports on Intel Skylake CPU and later. This is due to a known performance issue when using P2P on these CPU versions. There is now a new BIOS and performance tuning option available (PCIe Peer-to-Peer Serialization) from Intel and their OEM vendors that resolves

this P2P bandwidth issue. If the BIOS performance tuning option has been enabled, then NCCL direct P2P connections can be re-enabled by setting `NCCL_P2P_LEVEL=5`.

Chapter 10. NCCL Release 2.4.7

This is the NCCL 2.4.7 release notes. This release includes fixes from the previous NCCL 2.4.x releases as well as the following additional changes. For previous NCCL release notes, see the archived NCCL Release Notes.

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Improved bootstrap socket connection reliability at scale.
- Added detection of IBM/Power NVLink bridge device.
- Added NUMA support to PCI-E distance calculations on x86 architectures.

> **Note:** This adds a new level (5) for the `NCCL_P2P_LEVEL` and `NCCL_NET_GDR_LEVEL` environment variables. See the NCCL documentation for more details.

- Added the `NCCL_IGNORE_CPU_AFFINITY` environment variable.

Compatibility

NCCL 2.4.7 has been tested with the following:

- Deep learning framework 19.04 containers
- This NCCL release supports; CUDA 9.0, CUDA 9.2 , CUDA 10.0, and CUDA 10.1.

Fixed Issues

The following issues have been resolved in NCCL 2.4.7:

- Fixed hostname hashing issue. (GitHub issue #187)
- Fixed memory leaks. (GitHub issue #180)
- Fixed compiler warning. (GitHub issue #178)
- Replaced non-standard variable length arrays. (GitHub issue #171)
- Fixed Tree and Shared Memory crash. (GitHub PR #185)
- Fixed hangs during long running jobs.
- Fixed the `NCCL_RINGS` environment variable handling.

- Added extra checks to catch duplicate calls to `ncclCommDestroy()`. (GitHub issue #191)

Known Issues

- On single node Power systems with 4 GPUs, some performance regressions have been observed compared to NCCL 2.4.2. These will be addressed in future NCCL releases.
- By default, NCCL does not enable direct P2P communication through different PCIe root ports on Intel Skylake CPU and later. This is due to a known performance issue when using P2P on these CPU versions. There is now a new BIOS and performance tuning option available (PCIe Peer-to-Peer Serialization) from Intel and their OEM vendors that resolves this P2P bandwidth issue. If the BIOS performance tuning option has been enabled, then NCCL direct P2P connections can be re-enabled by setting `NCCL_P2P_LEVEL=5`.

Chapter 11. NCCL Release 2.4.2

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Implemented tree-based algorithms for better All Reduce performance at scale and with small and medium size messages.
- Support for external network plugins (e.g., `libfabric`).
- Add `ncclCommGetAsyncError()` function to report errors happening during collective operations.
- Add `ncclCommAbort()` function to destroy a communicator, aborting any outstanding operations.
- Support different ranks having a different `CUDA_VISIBLE_DEVICES`.
- Add a best-effort mechanism to check for size mismatch among collective calls.

Fixed Issues

- Support communication between Mesos containers (Github issue #155).
- Fix case where `posix_fallocate()` returns EINTR (Github issue #137).
- NCCL threads no longer escape the CPU affinity set by the user or job scheduler.

Chapter 12. NCCL Release 2.3.7

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Minor tuning of the LL threshold for multi-node jobs.
- Improve performance of initialization on large number of ranks.

Fixed Issues

- Fixed issue causing "WARN : Message truncated" errors.

Chapter 13. NCCL Release 2.3.5

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- This release is open-sourced with no new features or fixes.

Chapter 14. NCCL Release 2.3.4

Key Features and Enhancements

This NCCL release includes the following key features and enhancements.

- Improve performance tuning on large number of ranks.
- Add `NCCL_P2P_LEVEL` and `NCCL_IB_GDR_LEVEL` knobs to finely control when to use GPU Direct P2P and GPU Direct RDMA.
- Reduce setup time for large scale jobs.
- Increased maximum number of rings supported to 16.
- Added a runtime NCCL version API: `ncclGetVersion()`.
- Added NCCL_DEBUG_SUBSYS to allow filtering of `NCCL_DEBUG=INFO` logging from different subsystems.
- Support for Turing based systems.

Fixed Issues

- Fix hang on Power platforms.
- Fix low inter-node bandwidth issue on multi-DGX2 systems.
- Fix crash when used with PID isolator.

Chapter 15. 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 16. 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 17. 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 18. 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 19. 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 20. 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 21. 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 22. 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

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgment, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform 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 document. NVIDIA accepts no 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 document or (ii) customer product designs.

Trademarks