

[NCCL]([../index.html](#))

[2.25](<https://docs.nvidia.com/deeplearning/sdk/nccl-archived/index.html>)

- * [Overview of NCCL]([../overview.html](#))

- * [Setup]([../setup.html](#))

- * [Using NCCL]([../usage.html](#))

- * Creating a Communicator

- * Creating a communicator with options

- * Creating a communicator using multiple ncclUniquelds

- * Creating more communicators

- * Using multiple NCCL communicators concurrently

- * Finalizing a communicator

- * Destroying a communicator

- * Error handling and communicator abort

- * Asynchronous errors and error handling

- * Fault Tolerance

- * [Collective Operations]([collectives.html](#))

- * [AllReduce]([collectives.html#allreduce](#))

- * [Broadcast]([collectives.html#broadcast](#))

- * [Reduce]([collectives.html#reduce](#))

- * [AllGather]([collectives.html#allgather](#))

- * [ReduceScatter]([collectives.html#reducescatter](#))

- * [Data Pointers]([data.html](#))

- * [CUDA Stream Semantics]([streams.html](#))

- * [Mixing Multiple Streams within the same ncclGroupStart/End()

- group]([streams.html#mixing-multiple-streams-within-the-same-ncclgroupstart-end-group](#))

- * [\[Group Calls\]\(groups.html\)](#)
- * [\[Management Of Multiple GPUs From One Thread\]\(groups.html#management-of-multiple-gpus-from-one-thread\)](#)
- * [\[Aggregated Operations \(2.2 and later\)\]\(groups.html#aggregated-operations-2-2-and-later\)](#)
- * [\[Nonblocking Group Operation\]\(groups.html#nonblocking-group-operation\)](#)
- * [\[Point-to-point communication\]\(p2p.html\)](#)
- * [\[Sendrecv\]\(p2p.html#sendrecv\)](#)
- * [\[One-to-all \(scatter\)\]\(p2p.html#one-to-all-scatter\)](#)
- * [\[All-to-one \(gather\)\]\(p2p.html#all-to-one-gather\)](#)
- * [\[All-to-all\]\(p2p.html#all-to-all\)](#)
- * [\[Neighbor exchange\]\(p2p.html#neighbor-exchange\)](#)
- * [\[Thread Safety\]\(threadsafety.html\)](#)
- * [\[In-place Operations\]\(inplace.html\)](#)
- * [\[Using NCCL with CUDA Graphs\]\(cudagraph.html\)](#)
- * [\[User Buffer Registration\]\(bufferreg.html\)](#)
- * [\[NVLink Sharp Buffer Registration\]\(bufferreg.html#nvlink-sharp-buffer-registration\)](#)
- * [\[IB Sharp Buffer Registration\]\(bufferreg.html#ib-sharp-buffer-registration\)](#)
- * [\[General Buffer Registration\]\(bufferreg.html#general-buffer-registration\)](#)
- * [\[Memory Allocator\]\(bufferreg.html#memory-allocator\)](#)
- * [\[NCCL API\]\(../api.html\)](#)
- * [\[Communicator Creation and Management Functions\]\(../api/comms.html\)](#)
- * [\[ncclGetLastError\]\(../api/comms.html#ncclgetlasterror\)](#)
- * [\[ncclGetErrorString\]\(../api/comms.html#ncclgeterrorstring\)](#)
- * [\[ncclGetVersion\]\(../api/comms.html#ncclgetversion\)](#)
- * [\[ncclGetUniqueId\]\(../api/comms.html#ncclgetuniqueid\)](#)
- * [\[ncclCommInitRank\]\(../api/comms.html#ncclcomminitrank\)](#)
- * [\[ncclCommInitAll\]\(../api/comms.html#ncclcomminitall\)](#)

- * [\[ncclCommInitRankConfig\]\(../api/comms.html#ncclcomminitrankconfig\)](#)
- * [\[ncclCommInitRankScalable\]\(../api/comms.html#ncclcomminitrankscalable\)](#)
- * [\[ncclCommSplit\]\(../api/comms.html#ncclcommsplit\)](#)
- * [\[ncclCommFinalize\]\(../api/comms.html#ncclcommfinalize\)](#)
- * [\[ncclCommDestroy\]\(../api/comms.html#ncclcommdestroy\)](#)
- * [\[ncclCommAbort\]\(../api/comms.html#ncclcommabort\)](#)
- * [\[ncclCommGetAsyncError\]\(../api/comms.html#ncclcommgetasyncerror\)](#)
- * [\[ncclCommCount\]\(../api/comms.html#ncclcommcount\)](#)
- * [\[ncclCommCuDevice\]\(../api/comms.html#ncclcommcudevice\)](#)
- * [\[ncclCommUserRank\]\(../api/comms.html#ncclcommuserrank\)](#)
- * [\[ncclCommRegister\]\(../api/comms.html#ncclcommregister\)](#)
- * [\[ncclCommDeregister\]\(../api/comms.html#ncclcommderegister\)](#)
- * [\[ncclMemAlloc\]\(../api/comms.html#ncclmemalloc\)](#)
- * [\[ncclMemFree\]\(../api/comms.html#ncclmemfree\)](#)
- * [\[Collective Communication Functions\]\(../api/colls.html\)](#)
 - * [\[ncclAllReduce\]\(../api/colls.html#ncclallreduce\)](#)
 - * [\[ncclBroadcast\]\(../api/colls.html#ncclbroadcast\)](#)
 - * [\[ncclReduce\]\(../api/colls.html#ncclreduce\)](#)
 - * [\[ncclAllGather\]\(../api/colls.html#ncclallgather\)](#)
 - * [\[ncclReduceScatter\]\(../api/colls.html#ncclreducescatter\)](#)
- * [\[Group Calls\]\(../api/group.html\)](#)
 - * [\[ncclGroupStart\]\(../api/group.html#ncclgroupstart\)](#)
 - * [\[ncclGroupEnd\]\(../api/group.html#ncclgroupend\)](#)
 - * [\[ncclGroupSimulateEnd\]\(../api/group.html#ncclgroupsimulateend\)](#)
- * [\[Point To Point Communication Functions\]\(../api/p2p.html\)](#)
 - * [\[ncclSend\]\(../api/p2p.html#ncclsend\)](#)
 - * [\[ncclRecv\]\(../api/p2p.html#ncclrecv\)](#)

- * [\[Types\]\(../api/types.html\)](#)
- * [\[ncclComm_t\]\(../api/types.html#ncclcomm-t\)](#)
- * [\[ncclResult_t\]\(../api/types.html#ncclresult-t\)](#)
- * [\[ncclDataType_t\]\(../api/types.html#nccldatatype-t\)](#)
- * [\[ncclRedOp_t\]\(../api/types.html#ncclredop-t\)](#)
- * [\[ncclScalarResidence_t\]\(../api/types.html#ncclscalarresidence-t\)](#)
- * [\[ncclConfig_t\]\(../api/types.html#ncclconfig-t\)](#)
- * [\[ncclSimInfo_t\]\(../api/types.html#ncclsiminfo-t\)](#)
- * [\[User Defined Reduction Operators\]\(../api/ops.html\)](#)
- * [\[ncclRedOpCreatePreMulSum\]\(../api/ops.html#ncclredopcreatepremulsum\)](#)
- * [\[ncclRedOpDestroy\]\(../api/ops.html#ncclredopdestroy\)](#)
- * [\[Migrating from NCCL 1 to NCCL 2\]\(../nccl1.html\)](#)
- * [\[Initialization\]\(../nccl1.html#initialization\)](#)
- * [\[Communication\]\(../nccl1.html#communication\)](#)
- * [\[Counts\]\(../nccl1.html#counts\)](#)
- * [\[In-place usage for AllGather and ReduceScatter\]\(../nccl1.html#in-place-usage-for-allgather-and-reducescatter\)](#)
- * [\[AllGather arguments order\]\(../nccl1.html#allgather-arguments-order\)](#)
- * [\[Datatypes\]\(../nccl1.html#datatypes\)](#)
- * [\[Error codes\]\(../nccl1.html#error-codes\)](#)
- * [\[Examples\]\(../examples.html\)](#)
- * [\[Communicator Creation and Destruction Examples\]\(../examples.html#communicator-creation-and-destruction-examples\)](#)
- * [\[Example 1: Single Process, Single Thread, Multiple Devices\]\(../examples.html#example-1-single-process-single-thread-multiple-devices\)](#)
- * [\[Example 2: One Device per Process or Thread\]\(../examples.html#example-2-one-device-per-process-or-thread\)](#)

- * [Example 3: Multiple Devices per Thread](../examples.html#example-3-multiple-devices-per-thread)
- * [Example 4: Multiple communicators per device](../examples.html#example-4-multiple-communicators-per-device)
- * [Communication Examples](../examples.html#communication-examples)
 - * [Example 1: One Device per Process or Thread](../examples.html#example-1-one-device-per-process-or-thread)
 - * [Example 2: Multiple Devices per Thread](../examples.html#example-2-multiple-devices-per-thread)
- * [NCCL and MPI](../mpi.html)
 - * [API](../mpi.html#api)
 - * [Using multiple devices per process](../mpi.html#using-multiple-devices-per-process)
 - * [ReduceScatter operation](../mpi.html#reducescatter-operation)
 - * [Send and Receive counts](../mpi.html#send-and-receive-counts)
 - * [Other collectives and point-to-point operations](../mpi.html#other-collectives-and-point-to-point-operations)
 - * [In-place operations](../mpi.html#in-place-operations)
 - * [Using NCCL within an MPI Program](../mpi.html#using-nccl-within-an-mpi-program)
 - * [MPI Progress](../mpi.html#mpi-progress)
 - * [Inter-GPU Communication with CUDA-aware MPI](../mpi.html#inter-gpu-communication-with-cuda-aware-mpi)
 - * [Environment Variables](../env.html)
 - * [System configuration](../env.html#system-configuration)
 - * [NCCL_SOCKET_IFNAME](../env.html#nccl-socket-ifname)
 - * [Values accepted](../env.html#values-accepted)
 - * [NCCL_SOCKET_FAMILY](../env.html#nccl-socket-family)
 - * [Values accepted](../env.html#id2)

* [NCCL_SOCKET_RETRY_CNT](../env.html#nccl-socket-retry-cnt)

* [Values accepted](../env.html#id3)

* [NCCL_SOCKET_RETRY_SLEEP_MSEC](../env.html#nccl-socket-retry-sleep-msec)

* [Values accepted](../env.html#id4)

* [NCCL_SOCKET_NTHREADS](../env.html#nccl-socket-nthreads)

* [Values accepted](../env.html#id5)

* [NCCL_NSOCKS_PERTHREAD](../env.html#nccl-nsocks-perthread)

* [Values accepted](../env.html#id6)

* [NCCL_CROSS_NIC](../env.html#nccl-cross-nic)

* [Values accepted](../env.html#id7)

* [NCCL_IB_HCA](../env.html#nccl-ib-hca)

* [Values accepted](../env.html#id8)

* [NCCL_IB_TIMEOUT](../env.html#nccl-ib-timeout)

* [Values accepted](../env.html#id9)

* [NCCL_IB_RETRY_CNT](../env.html#nccl-ib-retry-cnt)

* [Values accepted](../env.html#id10)

* [NCCL_IB_GID_INDEX](../env.html#nccl-ib-gid-index)

* [Values accepted](../env.html#id11)

* [NCCL_IB_ADDR_FAMILY](../env.html#nccl-ib-addr-family)

* [Values accepted](../env.html#id12)

* [NCCL_IB_ADDR_RANGE](../env.html#nccl-ib-addr-range)

* [Values accepted](../env.html#id13)

* [NCCL_IB_ROCE_VERSION_NUM](../env.html#nccl-ib-roce-version-num)

* [Values accepted](../env.html#id14)

* [NCCL_IB_SL](../env.html#nccl-ib-sl)

* [Values accepted](../env.html#id15)

* [NCCL_IB_TC](../env.html#nccl-ib-tc)

* [Values accepted](../env.html#id16)

* [NCCL_IB_FIFO_TC](../env.html#nccl-ib-fifo-tc)

* [Values accepted](../env.html#id17)

* [NCCL_IB_RETURN_ASYNC_EVENTS](../env.html#nccl-ib-return-async-events)

* [Values accepted](../env.html#id18)

* [NCCL_OOB_NET_ENABLE](../env.html#nccl-oob-net-enable)

* [Values accepted](../env.html#id19)

* [NCCL_OOB_NET_IFNAME](../env.html#nccl-oob-net-ifname)

* [Values accepted](../env.html#id20)

* [NCCL_UID_STAGGER_THRESHOLD](../env.html#nccl-uid-stagger-threshold)

* [Values accepted](../env.html#id21)

* [NCCL_UID_STAGGER_RATE](../env.html#nccl-uid-stagger-rate)

* [Values accepted](../env.html#id22)

* [NCCL_NET](../env.html#nccl-net)

* [Values accepted](../env.html#id23)

* [NCCL_NET_PLUGIN](../env.html#nccl-net-plugin)

* [Values accepted](../env.html#id24)

* [NCCL_TUNER_PLUGIN](../env.html#nccl-tuner-plugin)

* [Values accepted](../env.html#id25)

* [NCCL_PROFILER_PLUGIN](../env.html#nccl-profiler-plugin)

* [Values accepted](../env.html#id26)

* [NCCL_IGNORE_CPU_AFFINITY](../env.html#nccl-ignore-cpu-affinity)

* [Values accepted](../env.html#id27)

* [NCCL_CONF_FILE](../env.html#nccl-conf-file)

* [Values accepted](../env.html#id28)

* [NCCL_DEBUG](../env.html#nccl-debug)

* [Values accepted](../env.html#id30)

- * [NCCL_DEBUG_FILE](../env.html#nccl-debug-file)
 - * [Values accepted](../env.html#id31)
- * [NCCL_DEBUG_SUBSYS](../env.html#nccl-debug-subsys)
 - * [Values accepted](../env.html#id32)
- * [NCCL_COLLNET_ENABLE](../env.html#nccl-collnet-enable)
 - * [Value accepted](../env.html#value-accepted)
- * [NCCL_COLLNET_NODE_THRESHOLD](../env.html#nccl-collnet-node-threshold)
 - * [Value accepted](../env.html#id33)
- * [NCCL_TOPO_FILE](../env.html#nccl-topo-file)
 - * [Value accepted](../env.html#id34)
- * [NCCL_TOPO_DUMP_FILE](../env.html#nccl-topo-dump-file)
 - * [Value accepted](../env.html#id35)
- * [NCCL_SET_THREAD_NAME](../env.html#nccl-set-thread-name)
 - * [Value accepted](../env.html#id36)
- * [Debugging](../env.html#debugging)
- * [NCCL_P2P_DISABLE](../env.html#nccl-p2p-disable)
 - * [Values accepted](../env.html#id37)
- * [NCCL_P2P_LEVEL](../env.html#nccl-p2p-level)
 - * [Values accepted](../env.html#id38)
 - * [Integer Values (Legacy)](../env.html#integer-values-legacy)
- * [NCCL_P2P_DIRECT_DISABLE](../env.html#nccl-p2p-direct-disable)
 - * [Values accepted](../env.html#id39)
- * [NCCL_SHM_DISABLE](../env.html#nccl-shm-disable)
 - * [Values accepted](../env.html#id40)
- * [NCCL_BUFFSIZE](../env.html#nccl-buffersize)
 - * [Values accepted](../env.html#id41)
- * [NCCL_NTHREADS](../env.html#nccl-nthreads)

* [Values accepted](../env.html#id42)

* [NCCL_MAX_NCHANNELS](../env.html#nccl-max-nchannels)

* [Values accepted](../env.html#id43)

* [NCCL_MIN_NCHANNELS](../env.html#nccl-min-nchannels)

* [Values accepted](../env.html#id44)

* [NCCL_CHECKS_DISABLE](../env.html#nccl-checks-disable)

* [Values accepted](../env.html#id45)

* [NCCL_CHECK_POINTERS](../env.html#nccl-check-pointers)

* [Values accepted](../env.html#id46)

* [NCCL_LAUNCH_MODE](../env.html#nccl-launch-mode)

* [Values accepted](../env.html#id47)

* [NCCL_IB_DISABLE](../env.html#nccl-ib-disable)

* [Values accepted](../env.html#id48)

* [NCCL_IB_AR_THRESHOLD](../env.html#nccl-ib-ar-threshold)

* [Values accepted](../env.html#id49)

* [NCCL_IB_QPS_PER_CONNECTION](../env.html#nccl-ib-qps-per-connection)

* [Values accepted](../env.html#id50)

* [NCCL_IB_SPLIT_DATA_ON_QPS](../env.html#nccl-ib-split-data-on-qps)

* [Values accepted](../env.html#id51)

* [NCCL_IB_CUDA_SUPPORT](../env.html#nccl-ib-cuda-support)

* [Values accepted](../env.html#id52)

* [NCCL_IB_PCI_RELAXED_ORDERING](../env.html#nccl-ib-pci-relaxed-ordering)

* [Values accepted](../env.html#id53)

* [NCCL_IB_ADAPTIVE_ROUTING](../env.html#nccl-ib-adaptive-routing)

* [Values accepted](../env.html#id54)

* [NCCL_IB_ECE_ENABLE](../env.html#nccl-ib-ece-enable)

* [Values accepted](../env.html#id55)

* [NCCL_MEM_SYNC_DOMAIN](../env.html#nccl-mem-sync-domain)

* [Values accepted](../env.html#id56)

* [NCCL_CUMEM_ENABLE](../env.html#nccl-cumem-enable)

* [Values accepted](../env.html#id57)

* [NCCL_CUMEM_HOST_ENABLE](../env.html#nccl-cumem-host-enable)

* [Values accepted](../env.html#id58)

* [NCCL_NET_GDR_LEVEL] (formerly

NCCL_IB_GDR_LEVEL)](../env.html#nccl-net-gdr-level-formerly-nccl-ib-gdr-level)

* [Values accepted](../env.html#id59)

* [Integer Values (Legacy)](../env.html#id60)

* [NCCL_NET_GDR_READ](../env.html#nccl-net-gdr-read)

* [Values accepted](../env.html#id61)

* [NCCL_NET_SHARED_BUFFERS](../env.html#nccl-net-shared-buffers)

* [Value accepted](../env.html#id62)

* [NCCL_NET_SHARED_COMMS](../env.html#nccl-net-shared-comms)

* [Value accepted](../env.html#id63)

* [NCCL_SINGLE_RING_THRESHOLD](../env.html#nccl-single-ring-threshold)

* [Values accepted](../env.html#id64)

* [NCCL_LL_THRESHOLD](../env.html#nccl-ll-threshold)

* [Values accepted](../env.html#id65)

* [NCCL_TREE_THRESHOLD](../env.html#nccl-tree-threshold)

* [Values accepted](../env.html#id66)

* [NCCL_ALGO](../env.html#nccl-algo)

* [Values accepted](../env.html#id67)

* [NCCL_PROTO](../env.html#nccl-proto)

* [Values accepted](../env.html#id68)

* [NCCL_NV_B_DISABLE](../env.html#nccl-nvb-disable)

* [Value accepted](../env.html#id69)

* [NCCL_PXN_DISABLE](../env.html#nccl-pxn-disable)

* [Value accepted](../env.html#id70)

* [NCCL_P2P_PXN_LEVEL](../env.html#nccl-p2p-pxn-level)

* [Value accepted](../env.html#id71)

* [NCCL_RUNTIME_CONNECT](../env.html#nccl-runtime-connect)

* [Value accepted](../env.html#id72)

* [NCCL_GRAPH_REGISTER](../env.html#nccl-graph-register)

* [Value accepted](../env.html#id74)

* [NCCL_LOCAL_REGISTER](../env.html#nccl-local-register)

* [Value accepted](../env.html#id75)

* [NCCL_LEGACY_CUDA_REGISTER](../env.html#nccl-legacy-cuda-register)

* [Value accepted](../env.html#id76)

* [NCCL_SET_STACK_SIZE](../env.html#nccl-set-stack-size)

* [Value accepted](../env.html#id77)

* [NCCL_GRAPH_MIXING_SUPPORT](../env.html#nccl-graph-mixing-support)

* [Value accepted](../env.html#id79)

* [NCCL_DMABUF_ENABLE](../env.html#nccl-dmabuf-enable)

* [Value accepted](../env.html#id80)

* [NCCL_P2P_NET_CHUNKSIZE](../env.html#nccl-p2p-net-chunksize)

* [Values accepted](../env.html#id81)

* [NCCL_P2P_LL_THRESHOLD](../env.html#nccl-p2p-ll-threshold)

* [Values accepted](../env.html#id82)

* [NCCL_ALLOC_P2P_NET_LL_BUFFERS](../env.html#nccl-alloc-p2p-net-ll-buffers)

* [Values accepted](../env.html#id83)

* [NCCL_COMM_BLOCKING](../env.html#nccl-comm-blocking)

* [Values accepted](../env.html#id84)

- * [\[NCCL_CGA_CLUSTER_SIZE\]\(../env.html#nccl-cga-cluster-size\)](#)
 - * [\[Values accepted\]\(../env.html#id85\)](#)
- * [\[NCCL_MAX_CTAS\]\(../env.html#nccl-max-ctas\)](#)
 - * [\[Values accepted\]\(../env.html#id86\)](#)
- * [\[NCCL_MIN_CTAS\]\(../env.html#nccl-min-ctas\)](#)
 - * [\[Values accepted\]\(../env.html#id87\)](#)
- * [\[NCCL_NVLS_ENABLE\]\(../env.html#nccl-nvls-enable\)](#)
 - * [\[Values accepted\]\(../env.html#id88\)](#)
- * [\[NCCL_IB_MERGE_NICS\]\(../env.html#nccl-ib-merge-nics\)](#)
 - * [\[Values accepted\]\(../env.html#id89\)](#)
- * [\[NCCL_MNNVL_ENABLE\]\(../env.html#nccl-mnnvl-enable\)](#)
 - * [\[Values accepted\]\(../env.html#id90\)](#)
- * [\[NCCL_RAS_ENABLE\]\(../env.html#nccl-ras-enable\)](#)
 - * [\[Values accepted\]\(../env.html#id91\)](#)
- * [\[NCCL_RAS_ADDR\]\(../env.html#nccl-ras-addr\)](#)
 - * [\[Values accepted\]\(../env.html#id92\)](#)
- * [\[NCCL_RAS_TIMEOUT_FACTOR\]\(../env.html#nccl-ras-timeout-factor\)](#)
 - * [\[Values accepted\]\(../env.html#id93\)](#)
- * [\[Troubleshooting\]\(../troubleshooting.html\)](#)
 - * [\[Errors\]\(../troubleshooting.html#errors\)](#)
 - * [\[RAS\]\(../troubleshooting.html#ras\)](#)
 - * [\[RAS\]\(../troubleshooting/ras.html\)](#)
 - * [\[Principle of Operation\]\(../troubleshooting/ras.html#principle-of-operation\)](#)
 - * [\[RAS Queries\]\(../troubleshooting/ras.html#ras-queries\)](#)
 - * [\[Sample Output\]\(../troubleshooting/ras.html#sample-output\)](#)
 - * [\[GPU Direct\]\(../troubleshooting.html#gpu-direct\)](#)
 - * [\[GPU-to-GPU communication\]\(../troubleshooting.html#gpu-to-gpu-communication\)](#)

- * [\[GPU-to-NIC communication\]\(../troubleshooting.html#gpu-to-nic-communication\)](#)
- * [\[PCI Access Control Services \(ACS\)\]\(../troubleshooting.html#pci-access-control-services-ac\)](#)
- * [\[Topology detection\]\(../troubleshooting.html#topology-detection\)](#)
- * [\[Shared memory\]\(../troubleshooting.html#shared-memory\)](#)
- * [\[Docker\]\(../troubleshooting.html#docker\)](#)
- * [\[Systemd\]\(../troubleshooting.html#systemd\)](#)
- * [\[Networking issues\]\(../troubleshooting.html#networking-issues\)](#)
- * [\[IP Network Interfaces\]\(../troubleshooting.html#ip-network-interfaces\)](#)
- * [\[IP Ports\]\(../troubleshooting.html#ip-ports\)](#)
- * [\[InfiniBand\]\(../troubleshooting.html#infiniband\)](#)

* [\[RDMA over Converged Ethernet](#)

[\(RoCE\)\]\(../troubleshooting.html#rdma-over-converged-ethernet-roce\)](#)

[__\[NCCL\]\(../index.html\)](#)

- * [\[Docs\]\(../index.html\)](#) »
- * [\[Using NCCL\]\(../usage.html\)](#) »
- * [Creating a Communicator](#)
- * [\[View page source\]\(../_sources/usage/communicators.rst.txt\)](#)

* * *

Creating a Communicator¶

When creating a communicator, a unique rank between 0 and n-1 has to be assigned to each of the n CUDA devices which are part of the communicator.

Using the same CUDA device multiple times as different ranks of the same NCCL

communicator is not supported and may lead to hangs.

Given a static mapping of ranks to CUDA devices, the

`[`ncclCommInitRank()`](../api/comms.html#c.ncclCommInitRank`

`"ncclCommInitRank"),`

`[`ncclCommInitRankConfig()`](../api/comms.html#c.ncclCommInitRankConfig`

`"ncclCommInitRankConfig")` and

`[`ncclCommInitAll()`](../api/comms.html#c.ncclCommInitAll "ncclCommInitAll")`

functions will create communicator objects, each communicator object being associated to a fixed rank and CUDA device. Those objects will then be used to launch communication operations.

Before calling `[`ncclCommInitRank()`](../api/comms.html#c.ncclCommInitRank`

`"ncclCommInitRank"),` you need to first create a unique object which will be

used by all processes and threads to synchronize and understand they are part of the same communicator. This is done by calling the

`[`ncclGetUniqueld()`](../api/comms.html#c.ncclGetUniqueld "ncclGetUniqueld")`

function.

The `[`ncclGetUniqueld()`](../api/comms.html#c.ncclGetUniqueld`

`"ncclGetUniqueld")` function returns an ID which has to be broadcast to all

participating threads and processes using any CPU communication system, for

example, passing the ID pointer to multiple threads, or broadcasting it to

other processes using MPI or another parallel environment using, for example, sockets.

You can also call the `ncclCommInitAll` operation to create `n` communicator

objects at once within a single process. As it is limited to a single process, this function does not permit inter-node communication. `ncclCommInitAll` is equivalent to calling a combination of `ncclGetUniqueId` and `ncclCommInitRank`.

The following sample code is a simplified implementation of `ncclCommInitAll`.

```
ncclResult_t ncclCommInitAll(ncclComm_t* comm, int ndev, const int* devlist) {  
    ncclUniqueId Id;  
    ncclGetUniqueId(&Id);  
    ncclGroupStart();  
    for (int i=0; i<ndev; i++) {  
        cudaSetDevice(devlist[i]);  
        ncclCommInitRank(comm+i, ndev, Id, i);  
    }  
    ncclGroupEnd();  
}
```

Related links:

```
> * [ `ncclCommInitAll()` ](../api/comms.html#c.ncclCommInitAll  
> "ncclCommInitAll")  
> * [ `ncclGetUniqueId()` ](../api/comms.html#c.ncclGetUniqueId  
> "ncclGetUniqueId")  
> * [ `ncclCommInitRank()` ](../api/comms.html#c.ncclCommInitRank
```

```
> "ncclCommInitRank")
```

```
>
```

```
## Creating a communicator with options
```

The `[ncclCommInitRankConfig()](../api/comms.html#c.ncclCommInitRankConfig "ncclCommInitRankConfig")` function allows to create a NCCL communicator with specific options.

The config parameters NCCL supports are listed here

`[ncclConfig_t](../api/types.html#ncclconfig)`.

For example, `blocking` can be set to 0 to ask NCCL to never block in any NCCL call, and at the same time other config parameters can be set as well to more precisely define communicator behavior. A simple example code is shown below:

```
ncclConfig_t config = NCCL_CONFIG_INITIALIZER;
config.blocking = 0;
config.minCTAs = 4;
config.maxCTAs = 16;
config.cgaClusterSize = 2;
config.netName = "Socket";
CHECK(ncclCommInitRankConfig(&comm, nranks, id, rank, &config));
do {
```



```

CHECK(ncclCommGetAsyncError(comm, &state));

// Handle outside events, timeouts, progress, ...

} while(state == ncclInProgress);

```

Related link:

[\[`ncclCommGetAsyncError`\]\(..../api/comms.html#c.ncclCommGetAsyncError
"ncclCommGetAsyncError"\)](#)

Creating a communicator using multiple ncclUniquesIds

The

[\[`ncclCommInitRankScalable`\]\(..../api/comms.html#c.ncclCommInitRankScalable
"ncclCommInitRankScalable"\)](#) function enables the creation of a NCCL

communicator using many ncclUniquesIds. All NCCL ranks have to provide the same array of ncclUniquesIds (same ncclUniquesIds, and in with the same order). For the best performance, we recommend distributing the ncclUniquesIds as evenly as possible amongst the NCCL ranks.

Internally, NCCL ranks will mostly communicate with a single ncclUniquesId.

Therefore, to obtain the best results, we recommend to evenly distribute ncclUniquesIds accross the ranks.

The following function can be used to decide if a NCCL rank should create a ncclUniquesIds:

```

bool rankHasRoot(const int rank, const int nRanks, const int nlds) {
    const int rmr = nRanks % nlds;
    const int rpr = nRanks / nlds;
    const int rlim = rmr * (rpr+1);
    if (rank < rlim) {
        return !(rank % (rpr + 1));
    } else {
        return !((rank - rlim) % rpr);
    }
}

```

For example, if 3 ncclUniquelds are to be distributed accross 7 NCCL ranks, the first ncclUniqueld will be associated to ranks 0-2, while the others will be associated to ranks 3-4, and 5-6. This function will therefore return true on rank 0, 3, and 5, and false otherwise.

Note: only the first ncclUniqueld will be used to create the communicator hash id, which is used to identify the communicator in the log file and in the replay tool.

Creating more communicators¶

The ncclCommSplit function can be used to create communicators based on an existing one. This allows to split an existing communicator into multiple sub-partitions, duplicate an existing communicator, or even create a single

communicator with fewer ranks.

The `ncclCommSplit` function needs to be called by all ranks in the original communicator. If some ranks will not be part of any sub-group, they still need to call `ncclCommSplit` with `color` being `NCCL_SPLIT_NOCOLOR`.

Newly created communicators will inherit the parent communicator configuration (e.g. non-blocking). If the parent communicator operates in non-blocking mode, a `ncclCommSplit` operation may be stopped by calling `ncclCommAbort` on the parent communicator, then on any new communicator returned. This is because a hang could happen during operations on any of the two communicators.

The following code duplicates an existing communicator:

```
int rank;  
  
ncclCommUserRank(comm, &rank);  
  
ncclCommSplit(comm, 0, rank, &newcomm, NULL);
```

This splits a communicator in two halves:

```
int rank, nranks;  
  
ncclCommUserRank(comm, &rank);
```

```
ncclCommCount(comm, &n ranks);
```

```
ncclCommSplit(comm, rank/(n ranks/2), rank%(n ranks/2), &newcomm, NULL);
```

This creates a communicator with only the first 2 ranks:

```
int rank;
```

```
ncclCommUserRank(comm, &rank);
```

```
ncclCommSplit(comm, rank<2 ? 0 : NCCL_SPLIT_NOCOLOR, rank, &newcomm, NULL);
```

Related links:

```
> * [ncclCommSplit()](../api/comms.html#c.ncclCommSplit "ncclCommSplit")
```

```
>
```

Using multiple NCCL communicators concurrently¶

Using multiple NCCL communicators requires careful synchronization, or can lead to deadlocks.

NCCL kernels are blocking (waiting for data to arrive), and any CUDA operation can cause a device synchronization, meaning it will wait for all NCCL kernels to complete. This can quickly lead to deadlocks since NCCL operations perform CUDA calls themselves.

Operations on different communicators should therefore be used at different epochs with a locking mechanism, and applications should ensure operations are submitted in the same order across ranks.

Launching multiple communication operations (on different streams) might work provided they can fit within the GPU, but could break at any time if NCCL were to use more CUDA blocks per operation, or if some calls used inside NCCL collectives were to perform a device synchronization (e.g. allocate some CUDA memory dynamically).

Finalizing a communicator¶

`ncclCommFinalize` will transition a communicator from the `_ncclSuccess_` state to the `_ncclInProgress_` state, start completing all operations in the background and synchronize with other ranks which may be using resources for their communications with other ranks. All uncompleted operations and network-related resources associated to a communicator will be flushed and freed with `ncclCommFinalize`. Once all NCCL operations are complete, the communicator will transition to the `_ncclSuccess_` state. Users can query that state with `ncclCommGetAsyncError`. If a communicator is marked as nonblocking, this operation is nonblocking; otherwise, it is blocking.

Related link: [`ncclCommFinalize()`](../api/comms.html#c.ncclCommFinalize "ncclCommFinalize")

Destroying a communicator¶

Once a communicator has been finalized, the next step is to free all resources, including the communicator itself. Local resources associated to a communicator can be destroyed with `ncclCommDestroy`. If the state of a communicator is `_ncclSuccess_` when calling `ncclCommDestroy`, the call is guaranteed to be nonblocking; otherwise `ncclCommDestroy` might block. In all cases, `ncclCommDestroy` call will free the resources of the communicator and return, and the communicator should no longer be accessed after `ncclCommDestroy` returns.

Related link: [`ncclCommDestroy()`](../api/comms.html#c.ncclCommDestroy "ncclCommDestroy")

Error handling and communicator abort¶

All NCCL calls return a NCCL error code which is summarized in the table below. If a NCCL call returns an error code different from `ncclSuccess` and `ncclInternalError`, and if `NCCL_DEBUG` is set to `WARN`, NCCL will print a human-readable message explaining what happened. If `NCCL_DEBUG` is set to `INFO`, NCCL will also print the call stack which led to the error. This message is intended to help the user fix the problem.

The table below summarizes how different errors should be understood and handled. Each case is explained in details in the following sections.

NCCL Errors¶				
Error	Description	Resolution	Error handling	Group behavior
---	---	---	---	---

ncclSuccess | No error | None | None | None

ncclUnhandledCudaError | Error during a CUDA call (1) | CUDA configuration / usage (1) | Communicator abort (5) | Global (6)

ncclSystemError | Error during a system call (1) | System configuration / usage (1) | Communicator abort (5) | Global (6)

ncclInternalError | Error inside NCCL (2) | Fix in NCCL (2) | Communicator abort (5) | Global (6)

ncclInvalidArgument | An argument to a NCCL call is invalid (3) | Fix in the application (3) | None (3) | Individual (3)

ncclInvalidUsage | The usage of NCCL calls is invalid (4) | Fix in the application (4) | Communicator abort (5) | Global (6)

ncclInProgress | The NCCL call is still in progress | Poll for completion using ncclCommGetAsyncError | None | None

(1) ncclUnhandledCudaError and ncclSystemError indicate that a call NCCL made to an external component failed, which caused the NCCL operation to fail. The error message should explain which component the user should look at and try to fix, potentially with the help of the administrators of the system.

(2) ncclInternalError denotes a NCCL bug. It might not report a message with NCCL_DEBUG=WARN since it requires a fix in the NCCL source code. NCCL_DEBUG=INFO will print the back trace which led to the error.

(3) ncclInvalidArgument indicates an argument value is incorrect, like a NULL pointer or an out-of-bounds value. When this error is returned, the NCCL call had no effect. The group state remains unchanged, the communicator is still functioning normally. The application can call ncclCommAbort or continue as if the call did not happen. This error will be returned immediately for a call

happening within a group and applies to that specific NCCL call. It will not be returned by `ncclGroupEnd` since `ncclGroupEnd` takes no argument.

(4) `ncclInvalidUsage` is returned when a dynamic condition causes a failure, which denotes an incorrect usage of the NCCL API.

(5) These errors are fatal for the communicator. To recover, the application needs to call `ncclCommAbort` on the communicator and re-create it.

(6) Dynamic errors for operations within a group are always reported by `ncclGroupEnd` and apply to all operations within the group, which may or may not have completed. The application must call `ncclCommAbort` on all communicators within the group.

Asynchronous errors and error handling

Some communication errors, and in particular network errors, are reported through the `ncclCommGetAsyncError` function. Operations experiencing an asynchronous error will usually not progress and never complete. When an asynchronous error happens, the operation should be aborted and the communicator destroyed using `ncclCommAbort`. When waiting for NCCL operations to complete, applications should call `ncclCommGetAsyncError` and destroy the communicator when an error happens.

The following code shows how to wait on NCCL operations and poll for asynchronous errors, instead of using `cudaStreamSynchronize`.


```

int ncclStreamSynchronize(cudaStream_t stream, ncclComm_t comm) {
    cudaError_t cudaErr;

    ncclResult_t ncclErr, ncclAsyncErr;

    while (1) {

        cudaErr = cudaStreamQuery(stream);

        if (cudaErr == cudaSuccess)

            return 0;

        if (cudaErr != cudaErrorNotReady) {

            printf("CUDA Error : cudaStreamQuery returned %d\n", cudaErr);

            return 1;

        }

        ncclErr = ncclCommGetAsyncError(comm, &ncclAsyncErr);

        if (ncclErr != ncclSuccess) {

            printf("NCCL Error : ncclCommGetAsyncError returned %d\n", ncclErr);

            return 1;

        }

        if (ncclAsyncErr != ncclSuccess) {

            // An asynchronous error happened. Stop the operation and destroy
            // the communicator

            ncclErr = ncclCommAbort(comm);

            if (ncclErr != ncclSuccess)

                printf("NCCL Error : ncclCommDestroy returned %d\n", ncclErr);

```

```

// Caller may abort or try to create a new communicator.

return 2;

}

// We might want to let other threads (including NCCL threads) use the CPU.
sched_yield();

}

}

```

Related links:

```

> * [ `ncclCommGetAsyncError()` ](../api/comms.html#c.ncclCommGetAsyncError
> "ncclCommGetAsyncError")
> * [ `ncclCommAbort()` ](../api/comms.html#c.ncclCommAbort "ncclCommAbort")
>

```

Fault Tolerance¶

NCCL provides a set of features to allow applications to recover from fatal errors such as a network failure, a node failure, or a process failure. When such an error happens, the application should be able to call `ncclCommAbort` on the communicator to free all resources, then create a new communicator to continue. All NCCL calls can be non-blocking to ensure `ncclCommAbort` can be called at any point, during initialization, communication or when finalizing the communicator.

To correctly abort, when any rank in a communicator fails (e.g., due to a segmentation fault), all other ranks need to call `_ncclCommAbort_` to abort their own NCCL communicator. Users can implement methods to decide when and whether to abort the communicators and restart the NCCL operation. Here is an example showing how to initialize and split a communicator in a non-blocking manner, allowing for an abort at any point:

```
bool globalFlag;

bool abortFlag = false;

ncclConfig_t config = NCCL_CONFIG_INITIALIZER;

config.blocking = 0;

CHECK(ncclCommInitRankConfig(&comm, nRanks, id, myRank, &config));

do {

    CHECK(ncclCommGetAsyncError(comm, &state));

} while(state == ncclInProgress && checkTimeout() != true);

if (checkTimeout() == true || state != ncclSuccess) abortFlag = true;

/* sync abortFlag among all healthy ranks. */
reportErrorGlobally(abortFlag, &globalFlag);

if (globalFlag) {

    /* time is out or initialization failed: every rank needs to abort and restart. */

    ncclCommAbort(comm);

    /* restart NCCL; this is a user implemented function, it might include
```

```

    * resource cleanup and ncclCommInitRankConfig() to create new communicators. */
    restartNCCL(&comm);
}

/* nonblocking communicator split. */
CHECK(ncclCommSplit(comm, color, key, &childComm, &config));
do {
    CHECK(ncclCommGetAsyncError(comm, &state));
} while(state == ncclInProgress && checkTimeout() != true);

if (checkTimeout() == true || state != ncclSuccess) abortFlag = true;

/* sync abortFlag among all healthy ranks. */
reportErrorGlobally(abortFlag, &globalFlag);

if (globalFlag) {
    ncclCommAbort(comm);

    /* if chilComm is not NCCL_COMM_NULL, user should abort child communicator
    * here as well for resource reclamation. */
    if (childComm != NCCL_COMM_NULL) ncclCommAbort(childComm);
    restartNCCL(&comm);
}

/* application workload */

```

The `_checkTimeout_` function needs to be provided by users to determine what is the longest time the application should wait for NCCL initialization;

likewise, users can apply other methods to detect errors besides a timeout function. Similar methods can be applied to NCCL finalization as well.

[Next]([collectives.html](#) "Collective Operations") [Previous]([../usage.html](#) "Using NCCL")

* * *

(C) Copyright 2020, NVIDIA Corporation

Built with [Sphinx](<http://sphinx-doc.org/>) using a [theme](https://github.com/rtfd/sphinx_rtd_theme) provided by [Read the Docs](<https://readthedocs.org>).