###################### Environment Variables ###################### NCCL has an extensive set of environment variables to tune for specific usage. Environment variables can also be set statically in /etc/nccl.conf (for an administrator to set system-wide values) or in ${NCCL_CONF_FILE} (since 2.23; see below). For example, those files could contain : .. code:: C NCCL_DEBUG=WARN NCCL_SOCKET_IFNAME==ens1f0 There are two categories of environment variables. Some are needed to make NCCL follow system-specific configuration, and can be kept in scripts and system configuration. Other parameters listed in the "Debugging" section should not be used in production nor retained in scripts, or only as workaround, and removed as soon as the issue is resolved. Keeping them set may result in sub-optimal behavior, crashes, or hangs. System configuration ==================== .. _NCCL_SOCKET_IFNAME: NCCL_SOCKET_IFNAME \---------- The ``NCCL_SOCKET_IFNAME`` variable specifies which IP interfaces to use for communication. Values accepted ^^^^^^^^^^^^^^^ Define to a list of prefixes to filter interfaces to be used by NCCL. Multiple prefixes can be provided, separated by the ``,`` symbol. Using the ``^`` symbol, NCCL will exclude interfaces starting with any prefix in that list. To match (or not) an exact interface name, begin the prefix string with the ``=`` character. Examples: ``eth`` : Use all interfaces starting with ``eth``, e.g. ``eth0``, ``eth1``, ... ``=eth0`` : Use only interface ``eth0`` ``=eth0,eth1`` : Use only interfaces ``eth0`` and ``eth1`` ``^docker`` : Do not use any interface starting with ``docker`` ``^=docker0`` : Do not use interface ``docker0``. Note: By default, the loopback interface (``lo``) and docker interfaces (``docker*``) would not be selected unless there are no other interfaces available. If you prefer to use ``lo`` or ``docker*`` over other interfaces, you would need to explicitly select them using ``NCCL_SOCKET_IFNAME``. The default algorithm will also favor interfaces starting with ``ib`` over others. Setting ``NCCL_SOCKET_IFNAME`` will bypass the automatic interface selection algorithm and may use all interfaces matching the manual selection. NCCL_SOCKET_FAMILY \---------- The ``NCCL_SOCKET_FAMILY`` variable allows users to force NCCL to use only IPv4 or IPv6 interface. Values accepted ^^^^^^^^^^^^^^^ Set to ``AF_INET`` to force the use of IPv4, or ``AF_INET6`` to force IPv6 usage. NCCL_SOCKET_RETRY_CNT \---------- (since 2.24) The

``NCCL_SOCKET_RETRY_CNT`` variable specifies the number of times NCCL retries to establish a socket connection after an ``ETIMEDOUT``, ``ECONNREFUSED``, or ``EHOSTUNREACH`` error. Values accepted ^^^^^^^^^^^^^^^ The default value is 34, any positive value is valid. NCCL_SOCKET_RETRY_SLEEP_MSEC \---------- (since 2.24) The ``NCCL_SOCKET_RETRY_SLEEP_MSEC`` variable specifies the number of milliseconds NCCL waits before retrying to establish a socket connection after the first ``ETIMEDOUT``, ``ECONNREFUSED``, or ``EHOSTUNREACH`` error. For subsequent errors, the waiting time scales linearly with the error count. The total time will therefore be (N+1) * N/2 * ``NCCL_SOCKET_RETRY_SLEEP_MSEC``, where N is given by ``NCCL_SOCKET_RETRY_CNT``. With the default values of ``NCCL_SOCKET_RETRY_CNT`` and ``NCCL_SOCKET_RETRY_SLEEP_MSEC``, the total retry time will be approx. 60 seconds. Values accepted ^^^^^^^^^^^^^^^ The default value is 100 milliseconds, any positive value is valid. NCCL_SOCKET_NTHREADS \---------- (since 2.4.8) The ``NCCL_SOCKET_NTHREADS`` variable specifies the number of CPU helper threads used per network connection for socket transport. Increasing this value may increase the socket transport performance, at the cost of a higher CPU usage. Values accepted ^^^^^^^^^^^^^^^ 1 to 16. On AWS, the default value is 2; on Google Cloud instances with the gVNIC network interface, the default value is 4 (since 2.5.6); in other cases, the default value is 1. For generic 100G networks, this value can be manually set to 4. However, the product of ``NCCL_SOCKET_NTHREADS`` and ``NCCL_NSOCKS_PERTHREAD`` cannot exceed 64. See also ``NCCL_NSOCKS_PERTHREAD``. NCCL_NSOCKS_PERTHREAD \---------- (since 2.4.8) The ``NCCL_NSOCKS_PERTHREAD`` variable specifies the number of sockets opened by each helper thread of the socket transport. In environments where per-socket speed is limited, setting this variable larger than 1 may improve the network performance. Values accepted ^^^^^^^^^^^^^^^ On AWS, the default value is 8; in other cases, the default value is 1. For generic 100G networks, this value can be manually set to 4. However, the product of ``NCCL_SOCKET_NTHREADS`` and ``NCCL_NSOCKS_PERTHREAD`` cannot exceed 64. See also ``NCCL_SOCKET_NTHREADS``. NCCL_CROSS_NIC \---------- The

``NCCL_CROSS_NIC`` variable controls whether NCCL should allow rings/trees to use different NICs, causing inter-node communication to use different NICs on different nodes. To maximize inter-node communication performance when using multiple NICs, NCCL tries to use the same NICs when communicating between nodes, to allow for a network design where each NIC on a node connects to a different network switch (network rail), and avoid any risk of traffic flow interference. The ``NCCL_CROSS_NIC`` setting is therefore dependent on the network topology, and in particular on whether the network fabric is rail-optimized or not. This has no effect on systems with only one NIC. Values accepted ^^^^^^^^^^^^^^^ 0: Always use the same NIC for the same ring/tree, to avoid crossing network rails. Suited for networks with per NIC switches (rails), with a slow inter-rail connection. Note that if the communicator does not contain the same GPUs on each node, NCCL may still need to communicate across NICs. 1: Allow the use of different NICs for the same ring/tree. This is suited for networks where all NICs from a node are connected to the same switch, hence trying to communicate across the same NICs does not help avoiding flow collisions. 2: (Default) Try to use the same NIC for the same ring/tree, but still allow for the use of different NICs if it would result in a better performance. NCCL_IB_HCA \---------- The ``NCCL_IB_HCA`` variable specifies which RDMA interfaces to use for communication. Values accepted ^^^^^^^^^^^^^^^ Define to filter IB Verbs interfaces to be used by NCCL. The list is comma-separated; port numbers can be specified using the ``:`` symbol. An optional prefix ``^`` indicates the list is an exclude list. A second optional prefix ``=`` indicates that the tokens are exact names, otherwise by default NCCL would treat each token as a prefix. Examples: ``mlx5`` : Use all ports of all cards starting with ``mlx5`` ``=mlx5_0:1,mlx5_1:1`` : Use ports 1 of cards ``mlx5_0`` and ``mlx5_1``. ``^=mlx5_1,mlx5_4`` : Do not use cards ``mlx5_1`` and ``mlx5_4``. Note: using ``mlx5_1`` without a preceding ``=`` will select ``mlx5_1`` as well as ``mlx5_10`` to ``mlx5_19``, if they exist. It is therefore always recommended to add the ``=`` prefix to ensure an exact match. NCCL_IB_TIMEOUT \---------- The ``NCCL_IB_TIMEOUT`` variable controls the InfiniBand Verbs Timeout. The timeout is computed as 4.096 Âµs * 2 ^ *timeout*, and the correct value is dependent on the size of the network. Increasing that value can help on very large networks, for

example, if NCCL is failing on a call to *ibv_poll_cq* with error 12. For more information, see section 12.7.34 of the InfiniBand specification Volume 1 (Local Ack Timeout). Values accepted ^^^^^^^^^^^^^^^ The default value used by NCCL is 20 (since 2.23; it was 18 since 2.14, and 14 before that). Values can be 1-31. NCCL_IB_RETRY_CNT \---------- (since 2.1.15) The ``NCCL_IB_RETRY_CNT`` variable controls the InfiniBand retry count. For more information, see section 12.7.38 of the InfiniBand specification Volume 1. Values accepted ^^^^^^^^^^^^^^^ The default value is 7. NCCL_IB_GID_INDEX \---------- (since 2.1.4) The ``NCCL_IB_GID_INDEX`` variable defines the Global ID index used in RoCE mode. See the InfiniBand *show_gids* command in order to set this value. For more information, see the InfiniBand specification Volume 1 or vendor documentation. Values accepted ^^^^^^^^^^^^^^^ The default value is -1. NCCL_IB_ADDR_FAMILY \---------- (since 2.21) The ``NCCL_IB_ADDR_FAMILY`` variable defines the IP address family associated to the infiniband GID dynamically selected by NCCL when ``NCCL_IB_GID_INDEX`` is left unset. Values accepted ^^^^^^^^^^^^^^^ The default value is "AF_INET". NCCL_IB_ADDR_RANGE \---------- (since 2.21) The ``NCCL_IB_ADDR_RANGE`` variable defines the range of valid GIDs dynamically selected by NCCL when ``NCCL_IB_GID_INDEX`` is left unset. Values accepted ^^^^^^^^^^^^^^^ By default, ignored if unset. GID ranges can be defined using the Classless Inter-Domain Routing (CIDR) format for IPv4 and IPv6 families. NCCL_IB_ROCE_VERSION_NUM \---------- (since 2.21) The ``NCCL_IB_ROCE_VERSION_NUM`` variable defines the RoCE version associated to the infiniband GID dynamically selected by NCCL when ``NCCL_IB_GID_INDEX`` is left unset. Values accepted ^^^^^^^^^^^^^^^ The default value is 2. NCCL_IB_SL \---------- (since 2.1.4) Defines the InfiniBand Service Level. For more information, see the InfiniBand specification Volume 1 or vendor documentation. Values accepted ^^^^^^^^^^^^^^^ The default value is 0. NCCL_IB_TC \---------- (since 2.1.15) Defines the InfiniBand traffic class field. For more information, see the InfiniBand specification Volume 1 or vendor documentation. Values accepted ^^^^^^^^^^^^^^^ The default value is 0. NCCL_IB_FIFO_TC \---------- (since 2.22.3) Defines the InfiniBand traffic class for control messages. Control messages are short RDMA write operations which control credit return, contrary

to other RDMA operations transmitting large segments of data. This setting allows to have those messages use a high priority, low-latency traffic class and avoid being delayed by the rest of the traffic. Values accepted ^^^^^^^^^^^^^^^ The default value is the traffic class set by NCCL_IB_TC, which defaults to 0 if not set. NCCL_IB_RETURN_ASYNC_EVENTS \---------- (since 2.23) IB events are reported to the user as warnings. If enabled, NCCL will also stop IB communications upon fatal IB asynchronous events. Values accepted ^^^^^^^^^^^^^^^ The default value is 1, set to 0 to disable NCCL_OOB_NET_ENABLE \---------- (since 2.23) The variable ``NCCL_OOB_NET_ENABLE`` enables the use of NCCL net for out-of-band communications. Enabling the usage of NCCL net will change the implementation of the allgather performed during the communicator initialization. Values accepted ^^^^^^^^^^^^^^^ Set the variable to 0 to disable, and to 1 to enable. NCCL_OOB_NET_IFNAME \---------- (since 2.23) If NCCL net is enabled for out-of-band communication (see ``NCCL_OOB_NET_ENABLE``), the ``NCCL_OOB_NET_IFNAME`` variable specifies which network interfaces to use. Values accepted ^^^^^^^^^^^^^^^ Define to filter interfaces to be used by NCCL for out-of-band communications. The list of accepted interface depends on the network used by NCCL. The list is comma-separated; port numbers can be specified using the ``:`` symbol. An optional prefix ``^`` indicates the list is an exclude list. A second optional prefix ``=`` indicates that the tokens are exact names, otherwise by default NCCL would treat each token as a prefix. If multiple devices are specified, NCCL will select the first matching device in the list. Example: ``NCCL_NET="IB" NCCL_OOB_NET_ENABLE=1 NCCL_OOB_NET_IFNAME="=mlx5_1"`` will use the Infiniband NET, with the interface ``mlx5_1`` ``NCCL_NET="IB" NCCL_OOB_NET_ENABLE=1 NCCL_OOB_NET_IFNAME="mlx5_1"`` will use the Infiniband NET, with the first interface found in the list of ``mlx5_1``, ``mlx5_10``, ``mlx5_11``, etc. ``NCCL_NET="Socket" NCCL_OOB_NET_ENABLE=1 NCCL_OOB_NET_IFNAME="ens1"`` will use the socket NET, with the first interface found in the list of ``ens1f0``, ``ens1f1``, etc. NCCL_UID_STAGGER_THRESHOLD \---------- (since 2.23) The ``NCCL_UID_STAGGER_THRESHOLD`` variable is used to trigger staggering of communications between NCCL ranks and the ncclUniqueId in order to avoid overflowing the ncclUniqueId. If the

number of NCCL ranks communicating exceeds the specified threshold, the communications are staggered using the rank value (see NCCL_UID_STAGGER_RATE below). If the number of NCCL ranks per ncclUniqueId is smaller or equal to the threshold, no staggering is performed. For example, if we have 128 NCCL ranks, 1 ncclUniqueId, and a threshold at 64, staggering is performed. However, if 2 ncclUniqueIds are used with 128 NCCL ranks and a threshold at 64, no staggering is done. Values accepted ^^^^^^^^^^^^^^^ The value of ``NCCL_UID_STAGGER_THRESHOLD`` must be a strictly positive integer. If unspecified, the default value is 256. NCCL_UID_STAGGER_RATE \---------- (since 2.23) The ``NCCL_UID_STAGGER_RATE`` variable is used to define the message rate targeted when staggering the communications between NCCL ranks and the ncclUniqueId. If staggering is used (see NCCL_UID_STAGGER_THRESHOLD above), the message rate is used to compute the time a given NCCL rank has to wait. Values accepted ^^^^^^^^^^^^^^^ The value of ``NCCL_UID_STAGGER_RATE`` must be a strictly positive integer, expressed in messages/second. If unspecified, the default value is 7000. NCCL_NET \-------- (since 2.10) Forces NCCL to use a specific network, for example to make sure NCCL uses an external plugin and doesn't automatically fall back on the internal IB or Socket implementation. Setting this environment variable will override the ``netName`` configuration in all communicators (see :ref:`ncclConfig`); if not set (undefined), the network module will be determined by the configuration; if not passing configuration, NCCL will automatically choose the best network module. Values accepted ^^^^^^^^^^^^^^^ The value of NCCL_NET has to match exactly the name of the NCCL network used (case-insensitive). Internal network names are "IB" (generic IB verbs) and "Socket" (TCP/IP sockets). External network plugins define their own names. Default value is undefined. NCCL_NET_PLUGIN \---------- (since 2.11) Set it to either a suffix string or to a library name to choose among multiple NCCL net plugins. This setting will cause NCCL to look for the net plugin library using the following strategy: \- If NCCL_NET_PLUGIN is set, attempt loading the library with name specified by NCCL_NET_PLUGIN; \- If NCCL_NET_PLUGIN is set and previous failed, attempt loading libnccl-net-.so; \- If NCCL_NET_PLUGIN is not set, attempt loading libnccl-net.so; \-

If no plugin was found (neither user defined nor default), use internal network plugin. For example, setting ``NCCL_NET_PLUGIN=foo`` will cause NCCL to try load ``foo`` and, if ``foo`` cannot be found, ``libnccl-net-foo.so`` (provided that it exists on the system). Values accepted ^^^^^^^^^^^^^^^ Plugin suffix, plugin file name, or "none". NCCL_TUNER_PLUGIN \---------- Set it to either a suffix string or to a library name to choose among multiple NCCL tuner plugins. This setting will cause NCCL to look for the tuner plugin library using the following strategy: \- If NCCL_TUNER_PLUGIN is set, attempt loading the library with name specified by NCCL_TUNER_PLUGIN; \- If NCCL_TUNER_PLUGIN is set and previous failed, attempt loading libnccl-net-.so; \- If NCCL_TUNER_PLUGIN is not set, attempt loading libnccl-tuner.so; \- If no plugin was found look for the tuner symbols in the net plugin (refer to ``NCCL_NET_PLUGIN``); \- If no plugin was found (neither through NCCL_TUNER_PLUGIN nor NCCL_NET_PLUGIN), use internal tuner plugin. For example, setting ``NCCL_TUNER_PLUGIN=foo`` will cause NCCL to try load ``foo`` and, if ``foo`` cannot be found, ``libnccl-tuner-foo.so`` (provided that it exists on the system). Values accepted ^^^^^^^^^^^^^^^ Plugin suffix, plugin file name, or "none". NCCL_PROFILER_PLUGIN \---------- Set it to either a suffix string or to a library name to choose among multiple NCCL profiler plugins. This setting will cause NCCL to look for the profiler plugin library using the following strategy: \- If NCCL_PROFILER_PLUGIN is set, attempt loading the library with name specified by NCCL_PROFILER_PLUGIN; \- If NCCL_PROFILER_PLUGIN is set and previous failed, attempt loading libnccl-profiler-.so; \- If NCCL_PROFILER_PLUGIN is not set, attempt loading libnccl-profiler.so; \- If no plugin was found (neither user defined nor default), do not enable profiling. \- If NCCL_PROFILER_PLUGIN is set to ``STATIC_PLUGIN``, the plugin symbols are searched in the program binary. For example, setting ``NCCL_PROFILER_PLUGIN=foo`` will cause NCCL to try load ``foo`` and, if ``foo`` cannot be found, ``libnccl-profiler-foo.so`` (provided that it exists on the system). Values accepted ^^^^^^^^^^^^^^^ Plugin suffix, plugin file name, or "none". NCCL_IGNORE_CPU_AFFINITY \---------- (since 2.4.6) The ``NCCL_IGNORE_CPU_AFFINITY`` variable can be used to cause NCCL to ignore the job's supplied CPU affinity and instead use the GPU affinity only. Values

accepted ^^^^^^^^^^^^^^^^ The default is 0, set to 1 to cause NCCL to ignore the job's supplied CPU affinity. NCCL_CONF_FILE \---------- (since 2.23) The ``NCCL_CONF_FILE`` variable allows the user to specify a file with the static configuration. This does not accept the ``~`` character as part of the path; please convert to a relative or absolute path first. Values accepted ^^^^^^^^^^^^^^^^ If unset or if the version is prior to 2.23, NCCL uses .nccl.conf in the home directory if available. .. _NCCL_DEBUG: NCCL_DEBUG \---------- The ``NCCL_DEBUG`` variable controls the debug information that is displayed from NCCL. This variable is commonly used for debugging. Values accepted ^^^^^^^^^^^^^^^^ VERSION - Prints the NCCL version at the start of the program. WARN - Prints an explicit error message whenever any NCCL call errors out. INFO - Prints debug information TRACE - Prints replayable trace information on every call. NCCL_DEBUG_FILE \---------- (since 2.2.12) The ``NCCL_DEBUG_FILE`` variable directs the NCCL debug logging output to a file. The filename format can be set to *filename.%h.%p* where *%h* is replaced with the hostname and *%p* is replaced with the process PID. This does not accept the ``~`` character as part of the path, please convert to a relative or absolute path first. Values accepted ^^^^^^^^^^^^^^^^ The default output file is *stdout* unless this environment variable is set. Setting ``NCCL_DEBUG_FILE`` will cause NCCL to create and overwrite any previous files of that name. Note: If the filename is not unique across all the job processes, then the output may be lost or corrupted. NCCL_DEBUG_SUBSYS \---------- (since 2.3.4) The ``NCCL_DEBUG_SUBSYS`` variable allows the user to filter the ``NCCL_DEBUG=INFO`` output based on subsystems. The value should be a comma separated list of the subsystems to include in the NCCL debug log traces. Prefixing the subsystem name with â€˜^â€™ will disable the logging for that subsystem. Values accepted ^^^^^^^^^^^^^^^^ The default value is INIT,BOOTSTRAP,ENV. Supported subsystem names are INIT (stands for initialization), COLL (stands for collectives), P2P (stands for peer-to-peer), SHM (stands for shared memory), NET (stands for network), GRAPH (stands for topology detection and graph search), TUNING (stands for algorithm/protocol tuning), ENV (stands for environment settings), ALLOC (stands for memory allocations), CALL (standard for function calls), PROXY (stands for the proxy thread operations), NVLS (standard for NVLink SHARP),

BOOTSTRAP (stands for early initialization), REG (stands for memory registration), PROFILE (stands for coarse-grained profiling of initialization), RAS (stands for reliability, availability, and serviceability subsystem) and ALL (includes every subsystem). NCCL_COLLNET_ENABLE \---------- (since 2.6) Enable the use of the CollNet plugin. Value accepted ^^^^^^^^^^^^^ Default is 0, define and set to 1 to use the CollNet plugin. NCCL_COLLNET_NODE_THRESHOLD \---------- (since 2.9.9) A threshold for the number of nodes below which CollNet will not be enabled. Value accepted ^^^^^^^^^^^^^ Default is 2, define and set to an integer. NCCL_TOPO_FILE \---------- (since 2.6) Path to an XML file to load before detecting the topology. By default, NCCL will load ``/var/run/nvidia-topologyd/virtualTopology.xml`` if present. Value accepted ^^^^^^^^^^^^^ A path to an accessible file describing part or all of the topology. NCCL_TOPO_DUMP_FILE \---------- (since 2.6) Path to a file to dump the XML topology to after detection. Value accepted ^^^^^^^^^^^^^ A path to a file which will be created or overwritten. NCCL_SET_THREAD_NAME \---------- (since 2.12) Give more meaningful names to NCCL CPU threads to ease debugging and analysis. Value accepted ^^^^^^^^^^^^^ 0 or 1. Default is 0 (disabled). Debugging ========= These environment variables should be used with caution. New versions of NCCL could work differently and forcing them to a particular value will prevent NCCL from selecting the best setting automatically. They can therefore cause performance problems in the long term, or even break some functionality. They are fine to use for experiments, or to debug a problem, but should generally not be set for production code. NCCL_P2P_DISABLE \---------- The ``NCCL_P2P_DISABLE`` variable disables the peer to peer (P2P) transport, which uses CUDA direct access between GPUs, using NVLink or PCI. Values accepted ^^^^^^^^^^^^^^^ Define and set to 1 to disable direct GPU-to-GPU (P2P) communication. NCCL_P2P_LEVEL \---------- (since 2.3.4) The ``NCCL_P2P_LEVEL`` variable allows the user to finely control when to use the peer to peer (P2P) transport between GPUs. The level defines the maximum distance between GPUs where NCCL will use the P2P transport. A short string representing the path type should be used to specify the topographical cutoff for using the P2P transport. If this isn't specified, NCCL will attempt to optimally select a value based on the architecture and environment it's run in. Values accepted

^^^^^^^^^^^^^^^ \- LOC : Never use P2P (always disabled) \- NVL : Use P2P when GPUs are connected through NVLink \- PIX : Use P2P when GPUs are on the same PCI switch. \- PXB : Use P2P when GPUs are connected through PCI switches (potentially multiple hops). \- PHB : Use P2P when GPUs are on the same NUMA node. Traffic will go through the CPU. \- SYS : Use P2P between NUMA nodes, potentially crossing the SMP interconnect (e.g. QPI/UPI). Integer Values (Legacy) ^^^^^^^^^^^^^^^^^^^^^^^ There is also the option to declare ``NCCL_P2P_LEVEL`` as an integer corresponding to the path type. These numerical values were kept for retro-compatibility, for those who used numerical values before strings were allowed. Integer values are discouraged due to breaking changes in path types - the literal values can change over time. To avoid headaches debugging your configuration, use string identifiers. \- LOC : 0 \- PIX : 1 \- PXB : 2 \- PHB : 3 \- SYS : 4 Values greater than 4 will be interpreted as SYS. NVL is not supported using the legacy integer values. NCCL_P2P_DIRECT_DISABLE \---------- The ``NCCL_P2P_DIRECT_DISABLE`` variable forbids NCCL to directly access user buffers through P2P between GPUs of the same process. This is useful when user buffers are allocated with APIs which do not automatically make them accessible to other GPUs managed by the same process and with P2P access. Values accepted ^^^^^^^^^^^^^^^ Define and set to 1 to disable direct user buffer access across GPUs. NCCL_SHM_DISABLE \---------- The ``NCCL_SHM_DISABLE`` variable disables the Shared Memory (SHM) transports. SHM is used between devices when peer-to-peer cannot happen, therefore, host memory is used. NCCL will use the network (i.e. InfiniBand or IP sockets) to communicate between the CPU sockets when SHM is disabled. Values accepted ^^^^^^^^^^^^^^^ Define and set to 1 to disable communication through shared memory (SHM). NCCL_BUFFSIZE \---------- The ``NCCL_BUFFSIZE`` variable controls the size of the buffer used by NCCL when communicating data between pairs of GPUs. Use this variable if you encounter memory constraint issues when using NCCL or you think that a different buffer size would improve performance. Values accepted ^^^^^^^^^^^^^^^ The default is 4194304 (4 MiB). Values are integers, in bytes. The recommendation is to use powers of 2. For example, 1024 will give a 1KiB buffer. NCCL_NTHREADS \---------- The ``NCCL_NTHREADS`` variable sets the number of CUDA

threads per CUDA block. NCCL will launch one CUDA block per communication channel. Use this variable if you think your GPU clocks are low and you want to increase the number of threads. You can also use this variable to reduce the number of threads to decrease the GPU workload. Values accepted ^^^^^^^^^^^^^^^ The default is 512 for recent generation GPUs, and 256 for some older generations. The values allowed are 64, 128, 256 and 512. NCCL_MAX_NCHANNELS \---------- (NCCL_MAX_NRINGS since 2.0.5, NCCL_MAX_NCHANNELS since 2.5.0) The ``NCCL_MAX_NCHANNELS`` variable limits the number of channels NCCL can use. Reducing the number of channels also reduces the number of CUDA blocks used for communication, hence the impact on GPU computing resources. The old ``NCCL_MAX_NRINGS`` variable (used until 2.4) still works as an alias in newer versions but is ignored if ``NCCL_MAX_NCHANNELS`` is set. This environment variable has been superseded by ``NCCL_MAX_CTAS`` which can also be set programmatically using :ref:`ncclCommInitRankConfig`. Values accepted ^^^^^^^^^^^^^^^ Any value above or equal to 1. NCCL_MIN_NCHANNELS \---------- (NCCL_MIN_NRINGS since 2.2.0, NCCL_MIN_NCHANNELS since 2.5.0) The ``NCCL_MIN_NCHANNELS`` variable controls the minimum number of channels you want NCCL to use. Increasing the number of channels also increases the number of CUDA blocks NCCL uses, which may be useful to improve performance; however, it uses more CUDA compute resources. This is especially useful when using aggregated collectives on platforms where NCCL would usually only create one channel. The old ``NCCL_MIN_NRINGS`` variable (used until 2.4) still works as an alias in newer versions, but is ignored if ``NCCL_MIN_NCHANNELS`` is set. This environment variable has been superseded by ``NCCL_MIN_CTAS`` which can also be set programmatically using :ref:`ncclCommInitRankConfig`. Values accepted ^^^^^^^^^^^^^^^ The default is platform dependent. Set to an integer value, up to 12 (up to 2.2), 16 (2.3 and 2.4) or 32 (2.5 and later). NCCL_CHECKS_DISABLE \---------- (since 2.0.5, deprecated in 2.2.12) The ``NCCL_CHECKS_DISABLE`` variable can be used to disable argument checks on each collective call. Checks are useful during development but can increase the latency. They can be disabled to improve performance in production. Values accepted ^^^^^^^^^^^^^^^ The default is 0, set to 1 to disable checks. NCCL_CHECK_POINTERS

\---------- (since 2.2.12) The ``NCCL_CHECK_POINTERS`` variable enables checking of the CUDA memory pointers on each collective call. Checks are useful during development but can increase the latency. Values accepted ^^^^^^^^^^^^^^^ The default is 0, set to 1 to enable checking. Setting to 1 restores the original behavior of NCCL prior to 2.2.12. NCCL_LAUNCH_MODE \---------- (since 2.1.0) The ``NCCL_LAUNCH_MODE`` variable controls how NCCL launches CUDA kernels. Values accepted ^^^^^^^^^^^^^^^ The default value is PARALLEL. Setting is to GROUP will use cooperative groups (CUDA 9.0 and later) for processes managing more than one GPU. This is deprecated in 2.9 and may be removed in future versions. NCCL_IB_DISABLE \---------- The ``NCCL_IB_DISABLE`` variable prevents the IB/RoCE transport from being used by NCCL. Instead, NCCL will fall back to using IP sockets. Values accepted ^^^^^^^^^^^^^^^ Define and set to 1 to disable the use of InfiniBand Verbs for communication (and force another method, e.g. IP sockets). NCCL_IB_AR_THRESHOLD \---------- (since 2.6) Threshold above which we send InfiniBand data in a separate message which can leverage adaptive routing. Values accepted ^^^^^^^^^^^^^^^ Size in bytes, the default value is 8192. Setting it above NCCL_BUFFSIZE will disable the use of adaptive routing completely. NCCL_IB_QPS_PER_CONNECTION \---------- (since 2.10) Number of IB queue pairs to use for each connection between two ranks. This can be useful on multi-level fabrics which need multiple queue pairs to have good routing entropy. See ``NCCL_IB_SPLIT_DATA_ON_QPS`` for different ways to split data on multiple QPs, as it can affect performance. Values accepted ^^^^^^^^^^^^^^^ Number between 1 and 128, default is 1. NCCL_IB_SPLIT_DATA_ON_QPS \---------- (since 2.18) This parameter controls how we use the queue pairs when we create more than one. Set to 1 (split mode), each message will be split evenly on each queue pair. This may cause a visible latency degradation if many QPs are used. Set to 0 (round-robin mode), queue pairs will be used in round-robin mode for each message we send. Operations which do not send multiple messages will not use all QPs. Values accepted ^^^^^^^^^^^^^^^ 0 or 1. Default is 0 (since NCCL 2.20). Setting it to 1 will enable split mode (default in 2.18 and 2.19). NCCL_IB_CUDA_SUPPORT \---------- (removed in 2.4.0, see NCCL_NET_GDR_LEVEL)

The ``NCCL_IB_CUDA_SUPPORT`` variable is used to force or disable the usage of GPU Direct RDMA. By default, NCCL enables GPU Direct RDMA if the topology permits it. This variable can disable this behavior or force the usage of GPU Direct RDMA in all cases. Values accepted ^^^^^^^^^^^^^^^ Define and set to 0 to disable GPU Direct RDMA. Define and set to 1 to force the usage of GPU Direct RDMA. NCCL_IB_PCI_RELAXED_ORDERING \---------- (since 2.12) Enable the use of Relaxed Ordering for the IB Verbs transport. Relaxed Ordering can greatly help the performance of InfiniBand networks in virtualized environments. Values accepted ^^^^^^^^^^^^^^^ Set to 2 to automatically use Relaxed Ordering if available. Set to 1 to force the use of Relaxed Ordering and fail if not available. Set to 0 to disable the use of Relaxed Ordering. Default is 2. NCCL_IB_ADAPTIVE_ROUTING \---------- (since 2.16) Enable the use of Adaptive Routing capable data transfers for the IB Verbs transport. Adaptive routing can improve the performance of communications at scale. A system defined Adaptive Routing enabled SL has to be selected accordingly (cf. ``NCCL_IB_SL``). Values accepted ^^^^^^^^^^^^^^^ Enabled (1) by default on IB networks. Disabled (0) by default on RoCE networks. Set to 1 to force use of Adaptive Routing capable data transmission. NCCL_IB_ECE_ENABLE \---------- (since 2.23) Enable the use of Enhanced Connection Establishment (ECE) on IB Verbs networks. Values accepted ^^^^^^^^^^^^^^^ Enabled (1) by default. Set to 0 to disable use of ECE network capabilities. NCCL_MEM_SYNC_DOMAIN \---------- (since 2.16) Sets the default Memory Sync Domain for NCCL kernels (CUDA 12.0 & sm90 and later). Memory Sync Domains can help eliminate interference between the NCCL kernels and the application compute kernels, when they use different domains. Values accepted ^^^^^^^^^^^^^^^ Default value is ``cudaLaunchMemSyncDomainRemote`` (1). Currently supported values are 0 and 1. NCCL_CUMEM_ENABLE \---------- (since 2.18) Use CUDA cuMem* functions to allocate memory in NCCL. Values accepted ^^^^^^^^^^^^^^^ 0 or 1. Default is 0 in 2.18 (disabled); since 2.19 this feature is auto-enabled by default if the system supports it (NCCL_CUMEM_ENABLE can still be used to override the autodetection). NCCL_CUMEM_HOST_ENABLE \---------- (since 2.23) Use CUDA cuMem* functions to allocate host memory in NCCL. Values accepted

^^^^^^^^^^^^^^^ 0 or 1. Default is 0 in 2.23; since 2.24, default is 1 if CUDA driver >= 12.6 and CUDA runtime >= 12.2 NCCL_NET_GDR_LEVEL (formerly NCCL_IB_GDR_LEVEL) \---------- (since 2.3.4. In 2.4.0, NCCL_IB_GDR_LEVEL was renamed to NCCL_NET_GDR_LEVEL) The ``NCCL_NET_GDR_LEVEL`` variable allows the user to finely control when to use GPU Direct RDMA between a NIC and a GPU. The level defines the maximum distance between the NIC and the GPU. A string representing the path type should be used to specify the topographical cutoff for GpuDirect. If this isn't specified, NCCL will attempt to optimally select a value based on the architecture and environment it's run in. Values accepted ^^^^^^^^^^^^^^^ \- LOC : Never use GPU Direct RDMA (always disabled). \- PIX : Use GPU Direct RDMA when GPU and NIC are on the same PCI switch. \- PXB : Use GPU Direct RDMA when GPU and NIC are connected through PCI switches (potentially multiple hops). \- PHB : Use GPU Direct RDMA when GPU and NIC are on the same NUMA node. Traffic will go through the CPU. \- SYS : Use GPU Direct RDMA even across the SMP interconnect between NUMA nodes (e.g., QPI/UPI) (always enabled). Integer Values (Legacy) ^^^^^^^^^^^^^^^^^^^^^^ There is also the option to declare ``NCCL_NET_GDR_LEVEL`` as an integer corresponding to the path type. These numerical values were kept for retro-compatibility, for those who used numerical values before strings were allowed. Integer values are discouraged due to breaking changes in path types - the literal values can change over time. To avoid headaches debugging your configuration, use string identifiers. \- LOC : 0 \- PIX : 1 \- PXB : 2 \- PHB : 3 \- SYS : 4 Values greater than 4 will be interpreted as SYS. NCCL_NET_GDR_READ \---------- The ``NCCL_NET_GDR_READ`` variable enables GPU Direct RDMA when sending data as long as the GPU-NIC distance is within the distance specified by ``NCCL_NET_GDR_LEVEL``. Before 2.4.2, GDR read is disabled by default, i.e. when sending data, the data is first stored in CPU memory, then goes to the InfiniBand card. Since 2.4.2, GDR read is enabled by default for NVLink-based platforms. Note: Reading directly from GPU memory when sending data is known to be slightly slower than reading from CPU memory on some platforms, such as PCI-E. Values accepted ^^^^^^^^^^^^^^^ 0 or 1. Define and set to 1 to use GPU Direct RDMA to send data to the NIC directly (bypassing CPU). Before 2.4.2, the default value is 0

for all platforms. Since 2.4.2, the default value is 1 for NVLink-based platforms and 0 otherwise. NCCL_NET_SHARED_BUFFERS \---------- (since 2.8) Allows the usage of shared buffers for inter-node point-to-point communication. This will use a single large pool for all remote peers, having a constant memory usage instead of increasing linearly with the number of remote peers. Value accepted ^^^^^^^^^^^^^^ Default is 1 (enabled). Set to 0 to disable. NCCL_NET_SHARED_COMMS \---------- (since 2.12) Reuse the same connections in the context of PXN. This allows for message aggregation but can also decrease the entropy of network packets. Value accepted ^^^^^^^^^^^^^^ Default is 1 (enabled). Set to 0 to disable. NCCL_SINGLE_RING_THRESHOLD \---------- (since 2.1, removed in 2.3) The ``NCCL_SINGLE_RING_THRESHOLD`` variable sets the limit under which NCCL will only use one ring. This will limit bandwidth but improve latency. Values accepted ^^^^^^^^^^^^^^^ The default value is 262144 (256kB) on GPUs with compute capability 7 and above. Otherwise, the default value is 131072 (128kB). Values are integers, in bytes. NCCL_LL_THRESHOLD \---------- (since 2.1, removed in 2.5) The ``NCCL_LL_THRESHOLD`` variable sets the size limit under which NCCL uses low-latency algorithms. Values accepted ^^^^^^^^^^^^^^^ The default is 16384 (up to 2.2) or is dependent on the number of ranks (2.3 and later). Values are integers, in bytes. NCCL_TREE_THRESHOLD \---------- (since 2.4, removed in 2.5) The ``NCCL_TREE_THRESHOLD`` variable sets the size limit under which NCCL uses tree algorithms instead of rings. Values accepted ^^^^^^^^^^^^^^^ The default is dependent on the number of ranks. Values are integers, in bytes. NCCL_ALGO \---------- (since 2.5) The ``NCCL_ALGO`` variable defines which algorithms NCCL will use. Values accepted ^^^^^^^^^^^^^^ (since 2.5) Comma-separated list of algorithms (not case sensitive) among: +-------------+---------------+ | Version | Algorithm | +=============+===============+ | 2.5+ | Ring | +-------------+---------------+ | 2.5+ | Tree | +-------------+---------------+ | 2.5 to 2.13 | Collnet | +-------------+---------------+ | 2.14+ | CollnetChain | +-------------+---------------+ | 2.14+ | CollnetDirect | +-------------+---------------+ | 2.17+ | NVLS | +-------------+---------------+ | 2.18+ | NVLSTree | +-------------+---------------+ | 2.23+ | PAT | +-------------+---------------+ NVLS and NVLSTree enable NVLink SHARP offload. To specify

algorithms to exclude (instead of include), start the list with ``^``. (since 2.24) The accepted values are expanded to allow more flexibility, and parsing will issue a warning and fail if an unexpected token is found. Also, if ``ring`` is not specified as a valid algorithm then it will not implicitly fall back to ``ring`` if there is no other valid algorithm for the function. Instead, it will fail. The format is now a semicolon-separated list of pairs of function name and list of algorithms, where the function name is optional for the first entry. If not present, then it applies to all functions not later listed. A colon separates the function (when present) and the comma-separated list of algorithms. Also, if the first character of the comma-separated list of algorithms is a caret (``^``), then all the selections are inverted. For example, ``NCCL_ALGO="ring,collnetdirect;allreduce:tree,collnetdirect;broadcast:ring"`` Will enable ring and collnetdirect for all functions, then enable tree and collnetdirect for allreduce and ring for broadcast. And, ``NCCL_ALGO=allreduce:^tree`` will allow the default (all algorithms available) for all the functions except allreduce, which will have all algorithms available except tree. The default is unset, which causes NCCL to automatically choose the available algorithms based on the node topology and architecture. NCCL_PROTO \---------- (since 2.5) The ``NCCL_PROTO`` variable defines which protocol(s) NCCL will be allowed to use. Users are discouraged from setting this variable, with the exception of disabling a specific protocol in case a bug in NCCL is suspected. In particular, enabling LL128 on platforms that don't support it can lead to data corruption. Values accepted ^^^^^^^^^^^^^^ (since 2.5) Comma-separated list of protocols (not case sensitive) among: ``LL``, ``LL128``, and ``Simple``. To specify protocols to exclude (instead of to include), start the list with ``^``. The default behavior enables all supported algorithms: equivalent to ``LL,LL128,Simple`` on platforms which support LL128, and ``LL,Simple`` otherwise. (since 2.24) The accepted values are expanded to allow more flexibility, just as decribed for ``NCCL_ALGO`` above, allowing the user to specify protocols for each function. NCCL_NVB_DISABLE \---------- (since 2.11) Disable intra-node communication through NVLink via an intermediate GPU. Value accepted ^^^^^^^^^^^^^^ Default is 0, set to 1 to disable this mechanism. NCCL_PXN_DISABLE \---------- (since 2.12) Disable inter-node communication using a non-local NIC, using NVLink and an intermediate GPU.

Value accepted ^^^^^^^^^^^^^^^ Default is 0, set to 1 to disable this mechanism. NCCL_P2P_PXN_LEVEL \---------- (since 2.12) Control in which cases PXN is used for send/receive operations. Value accepted ^^^^^^^^^^^^^^^ A value of 0 will disable the use of PXN for send/receive. A value of 1 will enable the use of PXN when the NIC preferred by the destination is not accessible through PCI switches. A value of 2 (default) will cause PXN to always be used, even if the NIC is connected through PCI switches, storing data from all GPUs within the node on an intermediate GPU to maximize aggregation. NCCL_RUNTIME_CONNECT \---------- (since 2.22) Dynamically connect peers during runtime (e.g., calling `ncclAllreduce()`) instead of init stage. Value accepted ^^^^^^^^^^^^^^^ Default is 1, set to 0 to connect peers at init stage. .. _NCCL_GRAPH_REGISTER: NCCL_GRAPH_REGISTER \---------- (since 2.11) Enable user buffer registration when NCCL calls are captured by CUDA Graphs. Effective only when: (i) the CollNet algorithm is being used; (ii) all GPUs within a node have P2P access to each other; (iii) there is at most one GPU per process. User buffer registration may reduce the number of data copies between user buffers and the internal buffers of NCCL. The user buffers will be automatically de-registered when the CUDA Graphs are destroyed. Value accepted ^^^^^^^^^^^^^^^ 0 or 1. Default value is 1 (enabled). NCCL_LOCAL_REGISTER \---------- (since 2.19) Enable user local buffer registration when users explicitly call *ncclCommRegister*. Value accepted ^^^^^^^^^^^^^^^ 0 or 1. Default value is 1 (enabled). NCCL_LEGACY_CUDA_REGISTER \---------- (since 2.24) Cuda buffers allocated through *cudaMalloc* (and related memory allocators) are legacy buffers. Registering legacy buffer can cause implicit synchronization, which is unsafe and can possibly cause a hang for NCCL. NCCL disables legacy buffer registration by default, and users should move to cuMem-based memory allocators for buffer registration. Value accepted ^^^^^^^^^^^^^^^ 0 or 1. Default value is 0 (disabled). NCCL_SET_STACK_SIZE \---------- (since 2.9) Set CUDA kernel stack size to the maximum stack size amongst all NCCL kernels. It may avoid a CUDA memory reconfiguration on load. Set to 1 if you experience hang due to CUDA memory reconfiguration. Value accepted ^^^^^^^^^^^^^^^ 0 or 1. Default value is 0 (disabled). .. _NCCL_GRAPH_MIXING_SUPPORT: NCCL_GRAPH_MIXING_SUPPORT \----------

(since 2.13) Enable/disable support for multiple outstanding NCCL calls from parallel CUDA graphs or a CUDA graph and non-captured NCCL calls. NCCL calls are considered outstanding starting from their host-side launch (e.g., a call to `ncclAllreduce()` for non-captured calls or `cudaGraphLaunch()` for captured calls) and ending when the device kernel execution completes. With graph mixing support disabled, the following use cases are NOT supported: 1\. Using a NCCL communicator (or split-shared communicators) from parallel graph launches, where parallel means on different streams without dependencies that would serialize their execution. 2\. Launching a non-captured NCCL collective during an outstanding graph launch that uses the same communicator (or split-shared communicators), regardless of stream ordering. The ability to disable support is motivated by observed hangs in the CUDA launches when support is enabled and multiple ranks have work launched via cudaGraphLaunch from the same thread. Value accepted ^^^^^^^^^^^^^^ 0 or 1. Default is 1 (enabled). NCCL_DMABUF_ENABLE \---------- (since 2.13) Enable GPU Direct RDMA buffer registration using the Linux dma-buf subsystem. The Linux dma-buf subsystem allows GPU Direct RDMA capable NICs to read and write CUDA buffers directly without CPU involvement. Value accepted ^^^^^^^^^^^^^^ 0 or 1. Default value is 1 (enabled), but the feature is automatically disabled if the Linux kernel or the CUDA/NIC driver do not support it. NCCL_P2P_NET_CHUNKSIZE \---------- (since 2.14) The ``NCCL_P2P_NET_CHUNKSIZE`` controls the size of messages sent through the network for ncclSend/ncclRecv operations. Values accepted ^^^^^^^^^^^^^^^ The default is 131072 (128 K). Values are integers, in bytes. The recommendation is to use powers of 2, hence 262144 would be the next value. NCCL_P2P_LL_THRESHOLD \---------- (since 2.14) The ``NCCL_P2P_LL_THRESHOLD`` is the maximum message size that NCCL will use the LL protocol for P2P operations. Values accepted ^^^^^^^^^^^^^^^ Decimal number. Default is 16384. NCCL_ALLOC_P2P_NET_LL_BUFFERS \---------- (since 2.14) ``NCCL_ALLOC_P2P_NET_LL_BUFFERS`` instructs communicators to allocate dedicated LL buffers for all P2P network connections. This enables all ranks to use the LL protocol for latency-bound send and receive operations below ``NCCL_P2P_LL_THRESHOLD`` sizes.

Intranode P2P transfers always have dedicated LL buffers allocated. If running all-to-all workloads with high numbers of ranks, this will result in a high scaling memory overhead. Values accepted ^^^^^^^^^^^^^^^ 0 or 1. Default value is 0 (disabled). NCCL_COMM_BLOCKING \---------- (since 2.14) The ``NCCL_COMM_BLOCKING`` variable controls whether NCCL calls are allowed to block or not. This includes all calls to NCCL, including init/finalize functions, as well as communication functions which may also block due to the lazy initialization of connections for send/receive calls. Setting this environment variable will override the ``blocking`` configuration in all communicators (see :ref:`ncclConfig`); if not set (undefined), communicator behavior will be determined by the configuration; if not passing configuration, communicators are blocking. Values accepted ^^^^^^^^^^^^^^^ 0 or 1. 1 indicates blocking communicators, and 0 indicates nonblocking communicators. The default value is undefined. NCCL_CGA_CLUSTER_SIZE \---------- (since 2.16) Set CUDA Cooperative Group Array (CGA) cluster size. On sm90 and later we have an extra level of hierarchy where we can group together several blocks within the Grid, called Thread Block Clusters. Setting this to non-zero will cause NCCL to launch the communication kernels with the Cluster Dimension attribute set accordingly. Setting this environment variable will override the ``cgaClusterSize`` configuration in all communicators (see :ref:`ncclconfig`); if not set (undefined), CGA cluster size will be determined by the configuration; if not passing configuration, NCCL will automatically choose the best value. Values accepted ^^^^^^^^^^^^^^^ 0 to 8. Default value is undefined. NCCL_MAX_CTAS \---------- (since 2.17) Set the maximal number of CTAs the NCCL should use. Setting this environment variable will override the ``maxCTAs`` configuration in all communicators (see :ref:`ncclconfig`); if not set (undefined), maximal CTAs will be determined by the configuration; if not passing configuration, NCCL will automatically choose the best value. Values accepted ^^^^^^^^^^^^^^^ Set to a positive integer value up to 32. Default value is undefined. NCCL_MIN_CTAS \---------- (since 2.17) Set the minimal number of CTAs the NCCL should use. Setting this environment variable will override the ``minCTAs`` configuration in all communicators (see :ref:`ncclconfig`); if not set (undefined), minimal CTAs will be determined by the configuration; if not passing configuration, NCCL will automatically choose the best value. Values accepted

^^^^^^^^^^^^^^^ Set to a positive integer value up to 32. Default value is undefined. NCCL_NVLS_ENABLE \---------- (since 2.17) Enable the use of NVLink SHARP (NVLS). NVLink SHARP is available in third-generation NVSwitch systems (NVLink4) with Hopper and later GPU architectures, allowing collectives such as ``ncclAllReduce`` to be offloaded to the NVSwitch domain. NVLS will be disabled automatically on systems which do not support the feature. Values accepted ^^^^^^^^^^^^^^^ Default is automatic detection, define and set to 0 to disable use of NVLink SHARP. NCCL_IB_MERGE_NICS \---------- (since 2.20) Enable NCCL to combine dual-port IB NICs into a single logical network device. This allows NCCL to more easily aggregate dual-port NIC bandwidth. Values accepted ^^^^^^^^^^^^^^^ Default is 1 (enabled), define and set to 0 to disable NIC merging NCCL_MNNVL_ENABLE \---------- (since 2.21) Enable NCCL to use Multi-Node NVLink (MNNVL) when available. If the system or driver are not Multi-Node NVLink capable then MNNVL will automatically be disabled. This feature also requires NCCL CUMEM support (``NCCL_CUMEM_ENABLE``) to be enabled. MNNVL requires a fully configured and operational IMEX domain for all the nodes that form the NVLink domain. See the CUDA documentation for more details on IMEX domains. Values accepted ^^^^^^^^^^^^^^^ Default is automatic detection, define and set to 0 to disable MNNVL support. .. _env_NCCL_RAS_ENABLE: NCCL_RAS_ENABLE \---------- (since 2.24) Enable NCCL's reliability, availability, and serviceability (RAS) subsystem, which can be used to query the health of NCCL jobs during execution (see :doc:`troubleshooting/ras`). Values accepted ^^^^^^^^^^^^^^^ Default is 1 (enabled); define and set to 0 to disable RAS. .. _env_NCCL_RAS_ADDR: NCCL_RAS_ADDR \---------- (since 2.24) Specify the IP address and port number of a socket that the RAS subsystem will listen on for client connections. RAS can share this socket between multiple processes but that would not be desirable if multiple independent NCCL jobs share a single node (and if those jobs belong to different users, the OS will not allow the socket to be shared). In such cases, each job should be started with a different value (e.g., ``localhost:12345``, ``localhost:12346``, etc.). Since ``localhost`` is normally used, only those with access to the nodes where the job is running can connect to the socket. If desired, the address of an externally accessible network interface can be specified

instead, which will make RAS accessible from other nodes (such as a cluster's head node), but that has security implications that should be considered. Values accepted ^^^^^^^^^^^^^^^ Default is ``localhost:28028``. Either a host name or an IP address can be used for the first part; an IPv6 address needs to be enclosed in square brackets (e.g., ``[::1]``). NCCL_RAS_TIMEOUT_FACTOR \---------- (since 2.24) Specify the multiplier factor to apply to all the timeouts of the RAS subsystem. RAS relies on multiple timeouts, ranging from 5 to 60 seconds, to determine the state of the application and to maintain its internal communication, with complex interdependecies between different timeouts. This variable can be used to scale up all these timeouts in a safe, consistent manner, should any of the defaults turn out to be too small; e.g., if the NCCL application is subject to high-overhead debugging/tracing/etc., which makes its execution less predictable. If one wants to use the ``ncclras`` client in such circumstances, its timeout may need to be increased as well (or disabled). Values accepted ^^^^^^^^^^^^^^^ Default is 1; define and set to larger values to increase the timeouts.