    * [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)

* [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)

* [NCCL_SHM_DISABLE](../env.html#nccl-shm-disable)
  * [Values accepted](../env.html#id40)
* [NCCL_BUFFSIZE](../env.html#nccl-buffsize)
  * [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_NVB_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)

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

* * *

User Buffer RegistrationÂ¶

User Buffer Registration is a feature that allows NCCL to directly send/receive/operate data through the user buffer without extra internal copy (zero-copy). It can accelerate collectives and greatly reduce the resource usage (e.g. #channel usage). NCCL provides two ways to register user buffers; one is _CUDA Graph_ registration, and the other is _Local_ registration. NCCL requires that for all NCCL communication function calls (e.g., allreduce, sendrecv, and so on), if any rank in a communicator passes registered buffers to a NCCL communication function, all other ranks in the same communicator must pass their registered buffers; otherwise, mixing registered and non-registered buffers can result in undefined behavior.

NVLink Sharp Buffer RegistrationÂ¶

Since 2.19.x, NCCL supports user buffer registration for NVLink Sharp (NVLS); any NCCL collectives (e.g., allreduce) that support NVLS algorithm can utilize this feature.

To enable the _CUDA Graph_ based buffer registration for NVLS, users have to comply with several requirements:

> * The buffer is allocated through
> [`ncclMemAlloc()`](../api/comms.html#c.ncclMemAlloc "ncclMemAlloc") or a
> qualified allocator (see Memory Allocator).
> * The NCCL operation is launched on a stream captured by a CUDA graph for
> each rank.

> * Offset to the head address of the buffer is the same in collectives for

> each rank.

>

Registered buffers will be deregistered when the CUDA graph is destroyed. Here

is a CUDA graph based buffer registration example:

```
    void* sendbuff;
    void* recvbuff;
    size_t count = 1 << 25;
    CHECK(ncclMemAlloc(&sendbuff, count * sizeof(float)));
    CHECK(ncclMemAlloc(&recvbuff, count * sizeof(float)));

    cudaGraph_t graph;
    CHECK(cudaStreamBeginCapture(stream, cudaStreamCaptureModeThreadLocal));
    CHECK(ncclAllReduce(sendbuff, recvbuff, 1024, ncclFloat, ncclSum, comm, stream));
    // Same offset to the sendbuff and recvbuff head address for each rank
      CHECK(ncclAllReduce((void*)((float*)sendbuff + 1024), (void*)((float*)recvbuff + 2048), 1024,
ncclFloat, ncclSum, comm, stream));
    CHECK(cudaStreamEndCapture(stream, &graph));

    cudaGraphExec_t instance;
    CHECK(cudaGraphInstantiate(&instance, graph, NULL, NULL, 0));
    CHECK(cudaGraphLaunch(instance, stream));
    CHECK(cudaStreamSynchronize(stream));
```

```
    CHECK(cudaGraphExecDestroy(instance));
    CHECK(cudaGraphDestroy(graph));

    CHECK(ncclMemFree(sendbuff));
    CHECK(ncclMemFree(recvbuff));
```

On the other hand, to enable the _Local_ based buffer registration for NVLS, users have to comply with the following requirements:

> * The buffer is allocated through
> [`ncclMemAlloc()`](../api/comms.html#c.ncclMemAlloc "ncclMemAlloc") or a
> qualified allocator (see Memory Allocator).
> * Register buffer with
> [`ncclCommRegister()`](../api/comms.html#c.ncclCommRegister
> "ncclCommRegister") before calling collectives for each rank.
> * Call NCCL collectives as usual but similarly keep the offset to the head
> address of the buffer the same for each rank.
>

Registered buffers will be deregistered when users explicitly call [`ncclCommDeregister()`](../api/comms.html#c.ncclCommDeregister "ncclCommDeregister"). Here is a local based buffer registration example:

```
    void* sendbuff;
```

```
    void* recvbuff;
    size_t count = 1 << 25;
    void* sendRegHandle;
    void* recvRegHandle;
    CHECK(ncclMemAlloc(&sendbuff, count * sizeof(float)));
    CHECK(ncclMemAlloc(&recvbuff, count * sizeof(float)));

    CHECK(ncclCommRegister(comm, sendbuff, count * sizeof(float), &sendRegHandle));
    CHECK(ncclCommRegister(comm, recvbuff, count * sizeof(float), &recvRegHandle));

    CHECK(ncclAllReduce(sendbuff, recvbuff, 1024, ncclFloat, ncclSum, comm, stream));
      CHECK(ncclAllReduce((void*)((float*)sendbuff + 1024), (void*)((float*)recvbuff + 2048), 1024,
ncclFloat, ncclSum, comm, stream));
    CHECK(cudaStreamSynchronize(stream));

    CHECK(ncclCommDeregister(comm, sendRegHandle));
    CHECK(ncclCommDeregister(comm, recvRegHandle));

    CHECK(ncclMemFree(sendbuff));
    CHECK(ncclMemFree(recvbuff));
```

For local based registration, users can register the buffer once at the beginning of the program and reuse the buffer multiple times to utilize registration benefits.

To save the memory, it is also valid to allocate a large chunk of buffer and

register it once. sendbuff and recvbuff can be further allocated through the big chunk for zero-copy NCCL operations as long as sendbuff and recvbuff satisfy the offset requirements. The following example shows a use case:

```
void* buffer;
void* handle;
void* sendbuff;
void* recvbuff;
size_t size = 1 << 29;

CHECK(ncclMemAlloc(&buffer, size));
CHECK(ncclCommRegister(comm, buffer, size, &handle));

// assign buffer chunk to sendbuff and recvbuff
sendbuff = buffer;
recvbuff = (void*)((uint8_t*)buffer + (1 << 20));

CHECK(ncclAllReduce(sendbuff, recvbuff, 1024, ncclFloat, ncclSum, comm, stream));
CHECK(cudaStreamSynchronize(stream));

CHECK(ncclCommDeregister(comm, handle));

CHECK(ncclMemFree(sendbuff));
```

IB Sharp Buffer RegistrationÂ¶

NCCL 2.21.x supports IB Sharp buffer registration, any NCCL collectives that support IB Sharp algorithm can benefit from the feature such as allreduce, reducescatter, and allgather. Currently, NCCL only supports IB Sharp buffer registration for the communicators which contain 1 rank per node, and the registration can reduce the number of NCCL SM usage down to 1.

To enable IB Sharp buffer registration by CUDA graph:

> * Allocate send and recv buffer with any CUDA allcator (e.g.,
> cudaMalloc/ncclMemAlloc)
> * Launch NCCL collectives with CUDA graph
>

To enable IB Sharp buffer registration by local registration:

> * Allocate send and recv buffer with any CUDA allcator (e.g.,
> cudaMalloc/ncclMemAlloc)
> * Register send and recv buffer for each rank in the communicator with
> ncclCommRegister
> * Launch NCCL collectives
>

General Buffer RegistrationÂ¶

Since 2.23.x, NCCL supports intra-node buffer registration, which targets all

peer-to-peer intra-node communications and brings less memory access, fewer SM usage and performance improvement. Either registering buffers by ncclCommRegister in the beginning or applying CUDA graph can enable intra-node buffer registration for NCCL collectives and sendrecv. The registered buffers can be allocated through legacy cuda API (e.g., cudaMalloc) as well as VMM API (e.g., cuMem* or ncclMemAlloc). However, VMM-allocated buffers are highly recommended since it is safer than legacy buffers during failure and abort.

Memory AllocatorÂ¶

For convenience, NCCL provides ncclMemAlloc function to help users to allocate buffers through VMM API, which can be used for NCCL registration later. It is only designed for NCCL so that it is not recommended to use ncclMemAlloc allocated buffers everywhere in the applications. For advanced users, if you want to create your own memory allocator for NVLS buffer registration, the allocator needs to satisfy the following requirements:

> * Allocate buffer with shared flag
> CU_MEM_HANDLE_TYPE_POSIX_FILE_DESCRIPTOR and also CU_MEM_HANDLE_TYPE_FABRIC
> on GPUs where itâ€™s supported.
> * Buffer size is multiple of multicast recommended granularity (i.e.
> cuMulticastGetGranularity(â€¦, CU_MULTICAST_GRANULARITY_RECOMMENDED))
> * Buffer head address is at least aligned to multicast minimal granularity
> (i.e. cuMulticastGetGranularity(â€¦, CU_MULTICAST_GRANULARITY_MINIMUM))
>

[Next](../api.html "NCCL API") [Previous](cudagraph.html "Using NCCL with CUDA Graphs")

* * *