NCCL Docs Overview

Thursday, August 7, 2025 3:17 PM

NCCL
- Communicators
  - Fault Tolerance
  - Error Handling
- Collective Operations
- CUDA Streams
- P2P
- Integration w/t cuda graph
- User Buffer Registration

(A) Collective Communication :-

* Receive Buffers + Output Buffer
* nccl ——→ AllReduce, Broadcast, Reduce, AllGather, ReduceScatter } API signature

* AllReduce = Reduce + Broadcast
= ReduceScatter + AllGather

* Data Pointers or how does NCCL interface w/t memory
  1) Accepts any CUDA pointer accessible on CUDA device linked to current communicator
  2) Direct Device memory
  3) Host memory registered to device
     ↳ CUDA SDK → cudaHostRegister
  4) Managed & unified memory

But it doesn't work with ROMA or GPUDirect based remote device memory.

NCCL_CHECK_POINTERS=1

Needs to be true for remote memory checks to apply.

* NCCL Streams → CUDA stream 0, CUDA stream 1 } Both streams blocked + Global sync bt streams.

NCCL Group ↑

Collective call once allocated to CUDA stream are run async wrt other ops within the stream.

(B) Group Calls :- Merge calls into 1.

→ Avoids collective launch overhead.
→ 1 thread ——→ Multiple GPUs (somehow avoids deadlock)
→ Create new P2P ops out of send & recv.

* Communicator init for rank can't be merged with other calls into a group.

* Managing multiple GPUs from the same thread :-
  ↳ Needs group semantics otherwise deadlock.

Note :
1) Init of comm should be in a group call but also the only collective op inside that group.
2) Enqueue in stream is not guaranteed inside a group call b/c by definition it will try to bundle for later call.
   ↳ Stream ops like cudaStreamSynchronize can only be called after group ends.

* Non-blocking Group call
  ↳ No guarantee that group is issued to CUDA stream.

ncclGroupEnd() ——→ ncclInProgress ⇒ still being issued
↳ ncclSuccess ⇒ NCCL kernel has been issued to stream

* Can't call CUDA stream calls until NCCL kernel is issued to stream.

(C) P2P

→ Calls blocking as a group but within a group they're independent.

* NCCL is not thread safe. Only 1 thread should interact w/t a communicator while others probe/poll status.

* For in-place NCCL ops simply have send Buffer = Recv Buffer

(D) CUDA GRAPHS :-

* NCCL 2.9 supports NCCL ops being captured in CUDA graphs.

CUDA Graphs { CUDA ops + NCCL ops — NCCL 2.9 (CUDA 11.3)

current version is 2.27 which is much higher

* Currently only CUDA ops are captured in CUDA graph.

```
cudaGraph_t graph;
cudaStreamBeginCapture(stream);
kernel_A<<< ..., stream >>>(...);
kernel_B<<< ..., stream >>>(...);
ncclAllreduce(..., stream);        → NCCL ops
kernel_C<<< ..., stream >>>(...);
cudaStreamEndCapture(stream, &graph);

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

(E) Zero-Copy

User buffer registration / zero-copy ops :-
↳ Either CUDA Graph Registration
↳ Local Registration

Prevents "internal" copy of data.
Must register from all ranks + sender buffer & receiver registration

* Types of user buffers →
  1) NVLink SHARP Buffers
  2) IB SHARP Buffers
  3) Symmetric Buffer Registration