Hewlett Packard
Enterprise

Enhancing PyTorch Performance on Frontier with the RCCL OFI-Plugin

Mengshiou Wu, Mark Stock

April 17th, 2024

Introduction

- RCCL (ROCm Communication Collectives Library) is used as the underlying communication library for major machine learning frameworks such as PyTorch or TensorFlow/Horovod.
- The default setting of RCCL uses TCP/IP for inter-node communications and does not utilize Slingshot/Libfabric.
- The Slurm scripts of OLCF's MLDL training usually include a few lines that enables RCCL to use a plugin to improve performance of MLDL codes:

#export NCCL_DEBUG=INFO
export LD_LIBRARY_PATH=${PATH TO THE PLUGIN}/libs/
#export FI_LOG_LEVEL=info
export NCCL_NET_GDR_LEVEL=3

Outline

- The focus of this talk is on the communication libraries used by PyTorch.
  - For PyTorch-DDP, PyTorch-FSDP, or introduction of different model parallelisms, OLCF has an excellent training talk (AI for Science at Scale – Part 2: https://github.com/olcf/ai-training-series).

- **Part I:**
  - The basic of RCCL, RCCL tester, and variables that may impact performance.
  - The basic of the OFI-Plugin.

- **Part II:**
  - PyTorch examples with and without using the Plugin.
  - Profiling and analyzing performance data using PyTorch Profiler.

Part I: RCCL, RCCL Tester and the Plugin

RCCL Basics

MLDL User Code (python, C++)

Framework (PyTorch, Tensorflow, Horovod)

Communication Library (MPI, **RCCL**, Gloo)

Transport layer (**libfabric**, IB, TCP/IP)

Other Network-y stuff (Socket)

Physical layers (xGMI, PCIe, Cat5, fiber)

RCCL Basics

- RCCL versions & library location
  - /opt/rocm-${version}/rccl/lib or /opt/rocm-${version}/rccl/include ➔ before 6.0.0
  - /opt/rocm-${version}/lib or /opt/rocm-${version}/include ➔ after 6.0.0
  - **Load the rocm module should be all you need to do.**

- Build RCCL from source
  - If plan to build RCCL from source: https://github.com/ROCm/rccl

- Most useful environment variables
  - **NCCL_NET_GDR_LEVEL=3** or **NCCL_NET_GDR_LEVEL=PHB** - This variable enables RDMA between GPUs.
  - **NCCL_ALGO** - Specify the algorithms for collective.
  - **NCCL_CROSS_NIC=1** - On large systems, this NCCL setting has been found to improve performance.
  - **NCCL_DEBUG** - For verification and debugging.

- RCCL is a port of NCCL and thus RCCL keeps some notations.

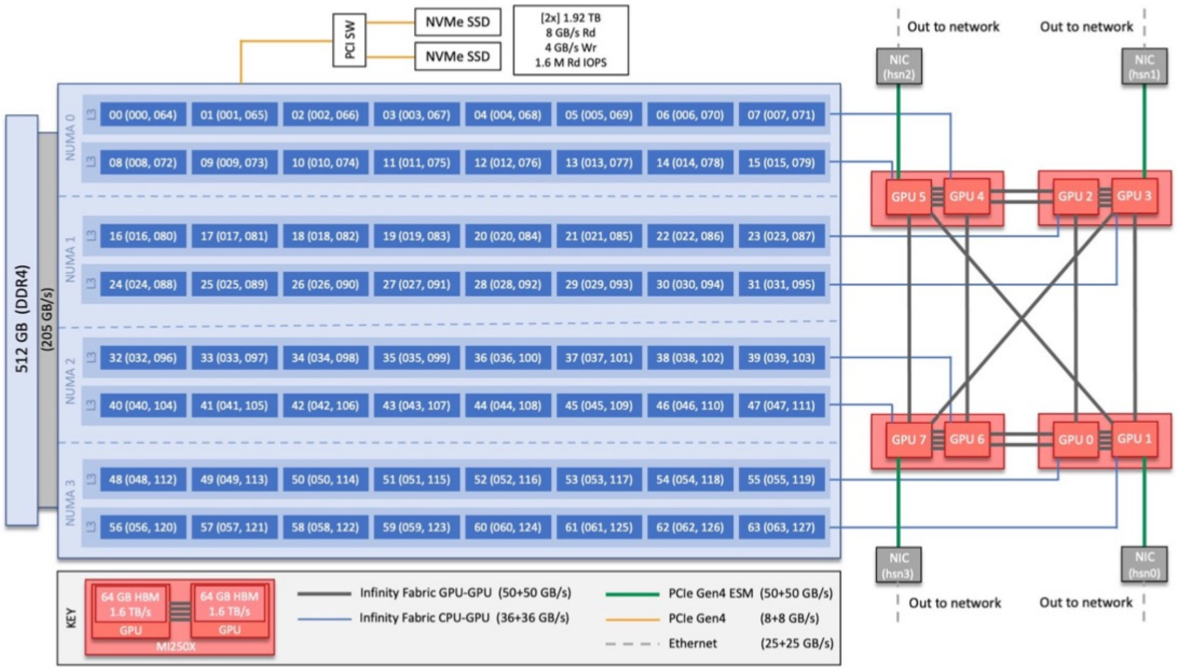
RCCL Basics

- Default intra-node communications
  - Utilize xGMI for collectives.

- Default inter-node communications on Frontier
  - TCP/IP: needs to use handshaking protocol, persistent connections.

- Collective Algorithms
  - Tree or Ring, depending on number of nodes/ranks and topology.
  - It uses complex tuning strategies rather than a simple threshold of message size.

Frontier Node Architecture



RCCL Performance

- https://github.com/ROCm/rccl-tests

- The tester use MPI only for summarized numbers; it does NOT use MPI for collectives.

- The tester can be used to test most collectives; we will focus on all_reduce here.

RCCL Tester Build Script

```
#!/usr/bin/bash -i
rocm_version=5.7.1

export LD_LIBRARY_PATH="${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}"
git clone --recursive https://github.com/ROCmSoftwarePlatform/rccl-tests

cd rccl-tests

grep -RiIl 'opt\/rocm' | xargs sed -i "s/opt\/rocm/opt\/rocm-$rocm_version/g"

module load libtool
module swap PrgEnv-cray PrgEnv-gnu
module load rocm/$rocm_version
module load craype-accel-amd-gfx90a
module load gcc/12.2.0
module load cray-mpich/8.1.27

echo $MPICH_DIR
make MPI=1 MPI_HOME=$MPICH_DIR ROCM_HOME=/opt/rocm-$rocm_version
export LD_LIBRARY_PATH=/opt/rocm-$rocm_version/rccl/lib:${LD_LIBRARY_PATH}

ldd ./build/all_reduce_perf
```

Check if the correct version of ROCm is selected.

RCCL Tester Definitions

- https://github.com/ROCm/rccl-tests/blob/develop/doc/PERFORMANCE.md
- **Algorithm bandwidth**
  - Algorithm bandwidth is using the most commonly used formula for bandwidth : **size (*S*) / time (*t*).** It is useful to compute how much time any large operation would take by simply dividing the size of the operation by the algorithm bandwidth.
  - algbw = S/t
- **Bus bandwidth**
  - This number is obtained **applying a multiple to the algorithm bandwidth to reflect the speed of the inter-GPU communication**. Using this bus bandwidth, we can compare it with the hardware peak bandwidth, independently of the number of ranks used.
    - AllReduce: algbw * (2*(n-1)/n)
    - ReduceScatter: algbw * (n-1)/n
    - AllGather:algbw * (n-1)/n
    - Broadcast: S/t
    - Reduce: S/t
- **In-Place:** use the same buffer for input/output.
- **Out-of-place:** use different buffers for input/output.

RCCL Test Data - Single Node

srun -n8 -N1 --tasks-per-node=8 --gpu-bind=closest all_reduce_perf -b 64K -e 4G -f 2 -g 1
(run setting: test message size from 64 K to 4G, double message size, use one GPU per rank)

#						out-of-place				in-place		
# size (B)	count (elements)	type	redop	root	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong
65536	16384	float	sum	-1	37.54	1.75	3.06	0	34.53	1.90	3.32	0
131072	32768	float	sum	-1	34.76	3.77	6.60	0	34.43	3.81	6.66	0
262144	65536	float	sum	-1	67.83	3.86	6.76	0	66.68	3.93	6.88	0
524288	131072	float	sum	-1	90.54	5.79	10.13	0	90.54	5.79	10.13	0
1048576	262144	float	sum	-1	101.7	10.31	18.05	0	101.4	10.34	18.10	0
2097152	524288	float	sum	-1	125.1	16.77	29.34	0	125.0	16.78	29.37	0
4194304	1048576	float	sum	-1	170.8	24.56	42.99	0	170.0	24.68	43.19	0
8388608	2097152	float	sum	-1	271.6	30.89	54.06	0	271.8	30.86	54.01	0
16777216	4194304	float	sum	-1	286.7	58.52	102.41	0	284.6	58.96	103.18	0
33554432	8388608	float	sum	-1	498.5	67.31	117.80	0	498.3	67.34	117.84	0
67108864	16777216	float	sum	-1	931.1	72.08	126.13	0	932.9	71.93	125.89	0
134217728	33554432	float	sum	-1	1804.6	74.38	130.16	0	1804.2	74.39	130.19	0
268435456	67108864	float	sum	-1	3552.8	75.56	132.22	0	3551.5	75.58	132.27	0
536870912	134217728	float	sum	-1	7059.0	76.05	133.10	0	7047.4	76.18	133.31	0
1073741824	268435456	float	sum	-1	13993	76.73	134.28	0	14009	76.64	134.13	0
2147483648	536870912	float	sum	-1	27847	77.12	134.96	0	27750	77.39	135.43	0
4294967296	1073741824	float	sum	-1	55157	77.87	136.27	0	55313	77.65	135.88	0

RCCL Test Data - 2 Nodes With TCP/IP for Inter Node Communications

srun -n16 -N2 --tasks-per-node=8 --gpu-bind=closest all_reduce_perf -b 64K -e 4G -f 2 -g 1

						out-of-place				in-place			
# size (B)	count (elements)	type	redop	root	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong
65536	16384	float	sum	-1	138.8	0.47	0.89	0	132.1	0.50	0.93	0
131072	32768	float	sum	-1	143.0	0.92	1.72	0	151.3	0.87	1.62	0
262144	65536	float	sum	-1	206.9	1.27	2.38	0	206.7	1.27	2.38	0
524288	131072	float	sum	-1	288.8	1.82	3.40	0	282.3	1.86	3.48	0
1048576	262144	float	sum	-1	438.2	2.39	4.49	0	447.1	2.35	4.40	0
2097152	524288	float	sum	-1	732.4	2.86	5.37	0	693.3	3.02	5.67	0
4194304	1048576	float	sum	-1	1235.0	3.40	6.37	0	1243.1	3.37	6.33	0
8388608	2097152	float	sum	-1	2086.3	4.02	7.54	0	2070.2	4.05	7.60	0
16777216	4194304	float	sum	-1	4069.6	4.12	7.73	0	3981.8	4.21	7.90	0
33554432	8388608	float	sum	-1	7602.9	4.41	8.28	0	7622.4	4.40	8.25	0
67108864	16777216	float	sum	-1	15715	4.27	8.01	0	15658	4.29	8.04	0
134217728	33554432	float	sum	-1	32255	4.16	7.80	0	32296	4.16	7.79	0
268435456	67108864	float	sum	-1	64577	4.16	7.79	0	64669	4.15	7.78	0
536870912	134217728	float	sum	-1	128849	4.17	7.81	0	128647	4.17	7.82	0
1073741824	268435456	float	sum	-1	252600	4.25	7.97	0	252861	4.25	7.96	0
2147483648	536870912	float	sum	-1	503080	4.27	8.00	0	503154	4.27	8.00	0
4294967296	1073741824	float	sum	-1	1002721	4.28	8.03	0	1003000	4.28	8.03	0

135.88 GB/s to 8.03 GB/s. No one is happy.

AWS-OFI-RCCL Basic

- From the plugin README: *"This project implements a plug-in which maps RCCLs connection-oriented transport APIs to* libfabric's *connection-less reliable interface."*

- This allows RCCL applications to take benefit of libfabric's transport layer services like reliable message support and operating system bypass.

- The current version of AWS-OFI-RCCL plugin implements NCCL network API (ncclNet_v5) to utilize Libfabric. For example, an isend in RCCL will use ofi_isend in the plugin instead.

- The current version of AWS-OFI-RCCL plugin does not implement collNet (for inter-node allReduce):
  - Collective algorithms are implemented in RCCL. No Libfabric/CXI involved.

- Read more about NCCL/RCCL Net Plugin in: https://github.com/ROCm/rccl/tree/develop/ext-net.

AWS-OFI-RCCL Plugin Build Script

```
#!/usr/bin/bash -i
rocm_version=5.7.1

git clone --recursive –depth=1 https://github.com/ROCmSoftwarePlatform/aws-ofi-rccl

cd aws-ofi-rccl

module load libtool
module swap PrgEnv-cray PrgEnv-gnu
module load rocm/$rocm_version
module load craype-accel-amd-gfx90a
module load gcc/12.2.0
module load cray-mpich/8.1.27

libfabric_path=/opt/cray/libfabric/1.15.2.0

./autogen.sh
export LD_LIBRARY_PATH=/opt/rocm-$rocm_version/hip/lib:$LD_LIBRARY_PATH

CC=cc CFLAGS=-I/opt/rocm-$rocm_version/rccl/include ./configure \
  --with-libfabric=$libfabric_path --with-rccl=/opt/rocm-$rocm_version --enable-trace \
  --prefix=$PWD --with-hip=/opt/rocm-$rocm_version/hip --with-mpi=$MPICH_DIR

make
make install
```

RCCL Test Data - 2 Nodes with the Plugin

LD_LIBRARY_PATH=${PATH TO THE PLUGIN DIRECTORY}:$LD_LIBRARY_PATH srun -n16 -N2 --tasks-per-node=8 --gpu-bind=closest all_reduce_perf -b 64K -e 4G -f 2 -g 1

# size (B)	count (elements)	type	redop	root	out-of-place time (us)	out-of-place algbw (GB/s)	out-of-place busbw (GB/s)	out-of-place #wrong	in-place time (us)	in-place algbw (GB/s)	in-place busbw (GB/s)	in-place #wrong
65536	16384	float	sum	-1	48.15	1.36	2.55	0	44.81	1.46	2.74	0
131072	32768	float	sum	-1	66.72	1.96	3.68	0	59.44	2.21	4.13	0
262144	65536	float	sum	-1	94.59	2.77	5.20	0	83.99	3.12	5.85	0
524288	131072	float	sum	-1	131.9	3.98	7.45	0	129.7	4.04	7.58	0
1048576	262144	float	sum	-1	151.5	6.92	12.98	0	151.0	6.94	13.02	0
2097152	524288	float	sum	-1	191.4	10.96	20.54	0	191.6	10.95	20.52	0
4194304	1048576	float	sum	-1	282.6	14.84	27.83	0	282.9	14.83	27.80	0
8388608	2097152	float	sum	-1	441.6	19.00	35.62	0	442.1	18.97	35.58	0
16777216	4194304	float	sum	-1	718.8	23.34	43.77	0	719.8	23.31	43.70	0
33554432	8388608	float	sum	-1	1502.7	22.33	41.87	0	1273.5	26.35	49.40	0
67108864	16777216	float	sum	-1	2436.5	27.54	51.64	0	2437.3	27.53	51.63	0
134217728	33554432	float	sum	-1	4617.0	29.07	54.51	0	4623.6	29.03	54.43	0
268435456	67108864	float	sum	-1	9119.3	29.44	55.19	0	9146.5	29.35	55.03	0
536870912	134217728	float	sum	-1	17938	29.93	56.12	0	17882	30.02	56.29	0
1073741824	268435456	float	sum	-1	35112	30.58	57.34	0	35057	30.63	57.43	0
2147483648	536870912	float	sum	-1	70499	30.46	57.11	0	70247	30.57	57.32	0
4294967296	1073741824	float	sum	-1	140733	30.52	57.22	0	141643	30.32	56.85	0

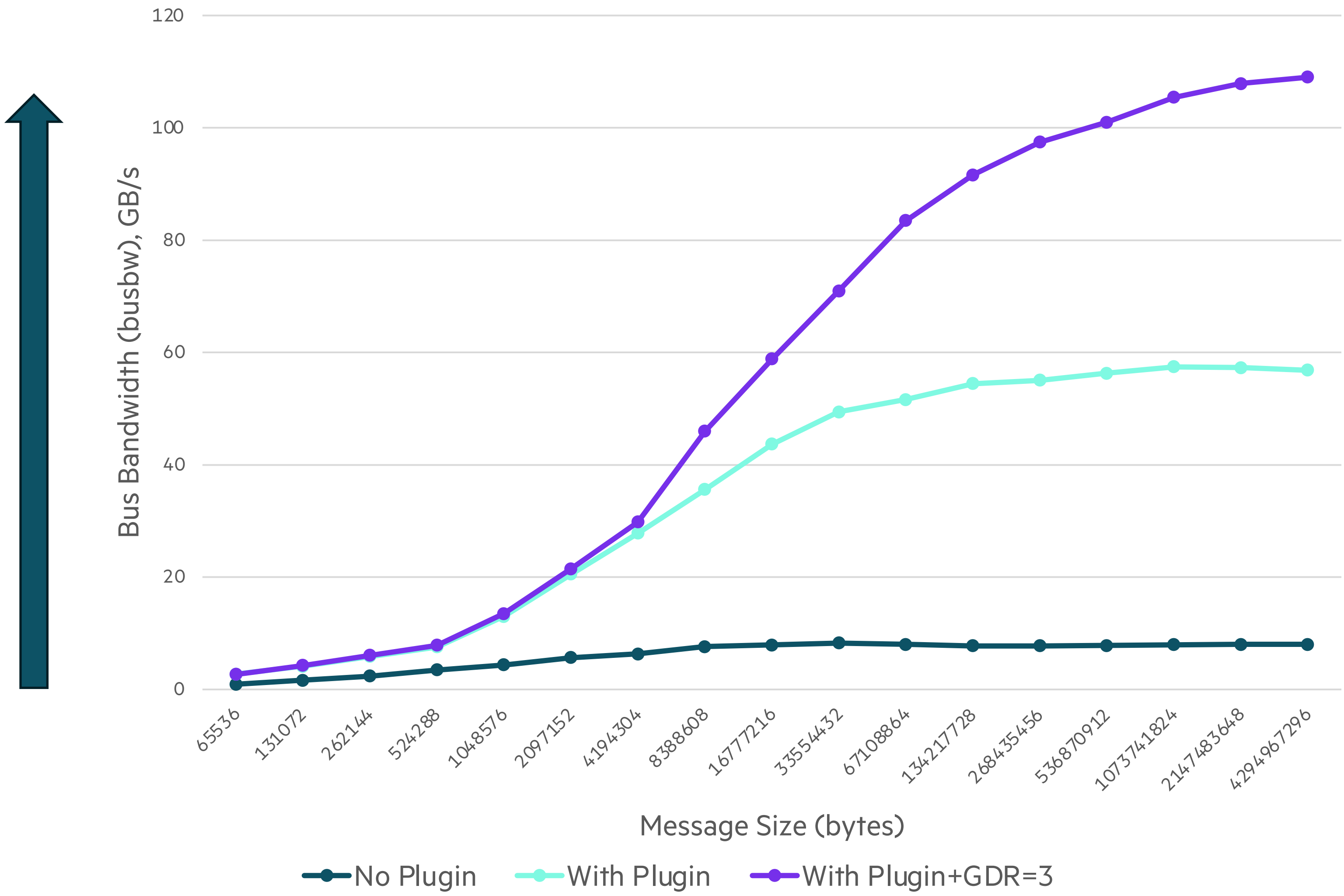
No need to rebuild your binaries.

RCCL Test Data - 2 Nodes with the Plugin + NCCL_NET_GDR_LEVEL=3

LD_LIBRARY_PATH=${PATH TO THE PLUGIN DIRECTORY}:$LD_LIBRARY_PATH
NCCL_NET_GDR_LEVEL=3 srun -n16 -N2 --tasks-per-node=8 --gpu-bind=closest all_reduce_perf -b 64K -e 4G -f 2 -g 1

						out-of-place				in-place		
# size (B)	count (elements)	type	redop	root	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong	time (us)	algbw (GB/s)	busbw (GB/s)	#wrong
65536	16384	float	sum	-1	47.80	1.37	2.57	0	45.96	1.43	2.67	0
131072	32768	float	sum	-1	66.79	1.96	3.68	0	57.10	2.30	4.30	0
262144	65536	float	sum	-1	87.64	2.99	5.61	0	81.01	3.24	6.07	0
524288	131072	float	sum	-1	125.5	4.18	7.83	0	125.2	4.19	7.85	0
1048576	262144	float	sum	-1	145.1	7.23	13.55	0	145.4	7.21	13.52	0
2097152	524288	float	sum	-1	181.7	11.54	21.64	0	183.1	11.46	21.48	0
4194304	1048576	float	sum	-1	266.1	15.76	29.56	0	263.6	15.91	29.83	0
8388608	2097152	float	sum	-1	341.0	24.60	46.13	0	342.0	24.53	45.99	0
16777216	4194304	float	sum	-1	549.4	30.54	57.26	0	534.5	31.39	58.85	0
33554432	8388608	float	sum	-1	1810.3	18.53	34.75	0	886.4	37.85	70.98	0
67108864	16777216	float	sum	-1	1508.5	44.49	83.41	0	1507.6	44.51	83.46	0
134217728	33554432	float	sum	-1	2747.7	48.85	91.59	0	2746.8	48.86	91.62	0
268435456	67108864	float	sum	-1	5165.4	51.97	97.44	0	5164.1	51.98	97.46	0
536870912	134217728	float	sum	-1	9971.1	53.84	100.95	0	9967.0	53.87	101.00	0
1073741824	268435456	float	sum	-1	19099	56.22	105.41	0	19098	56.22	105.42	0
2147483648	536870912	float	sum	-1	37328	57.53	107.87	0	37323	57.54	107.88	0
4294967296	1073741824	float	sum	-1	73869	58.14	109.02	0	73877	58.14	109.01	0

RCCL Test Data - All_reduce on 2 nodes, 16 ranks (n16N2)



Message Size	No Plugin	With Plugin	With Plugin+GDR=3
65536	0.93	2.74	2.67
131072	1.62	4.13	4.3
262144	2.38	5.85	6.07
524288	3.48	7.58	7.85
1048576	4.4	13.02	13.52
2097152	5.67	20.52	21.48
4194304	6.33	27.8	29.83
8388608	7.6	35.58	45.99
16777216	7.9	43.7	58.85
33554432	8.25	49.4	70.98
67108864	8.04	51.63	83.46
134217728	7.79	54.43	91.62
268435456	7.78	55.03	97.46
536870912	7.82	56.29	101
1073741824	7.96	57.43	105.42
2147483648	8	57.32	107.88
4294967296	8.03	56.85	109.01

Use NCCL_DEBUG=info to Verify Use of the Plugin

- Checking if the plugin exists:

```
frontier03329:113050:113050 [0] NCCL INFO NET/Plugin: Failed to find ncclNetPlugin_v6 symbol.
frontier03329:113050:113050 [0] NCCL INFO NET/Plugin: Loaded net plugin AWS Libfabric (v5)
frontier03329:113050:113050 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin_v6 symbol.
frontier03329:113050:113050 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin symbol (v4
or v5).
```

The plugin implements NCCL Net API v5

The plugin does not support ncclCollNet

- <u>Without the plugin</u>

  frontier01466:86382:86734 [0] NCCL INFO Channel 02/0 : 36[d9000] -> 45[de000] [send] via **NET/Socket**/4 comm 0x7ffbec000d70 nRanks 80

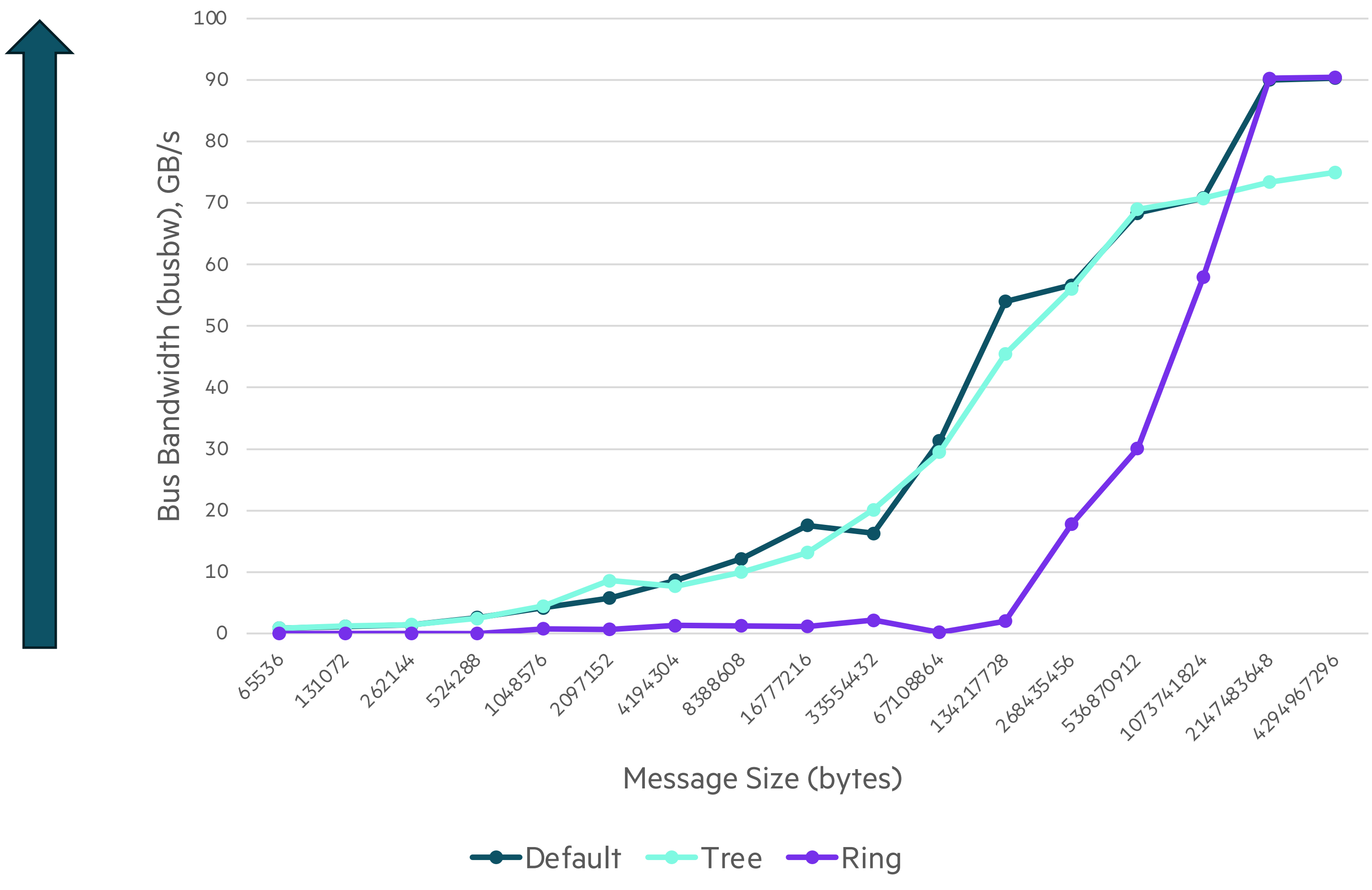
- <u>With the plugin</u>

  frontier08155:21778:21989 [0] NCCL INFO Channel 01/0 : 70[c1000] -> 74[c9000] [send] via **NET/AWS Libfabric**/0 comm 0x7ffbec000d70 nRanks 80

- <u>With the plugin + GDR3</u>

  frontier08431:43731:43989 [0] NCCL INFO Channel 01/0 : 14[c1000] -> 18[c9000] [send] via **NET/AWS Libfabric/**0**/GDRDMA** comm 0x7ffbec000d70 nRanks 80

Comparison of NCCL_ALGO, 64 nodes, 512 ranks



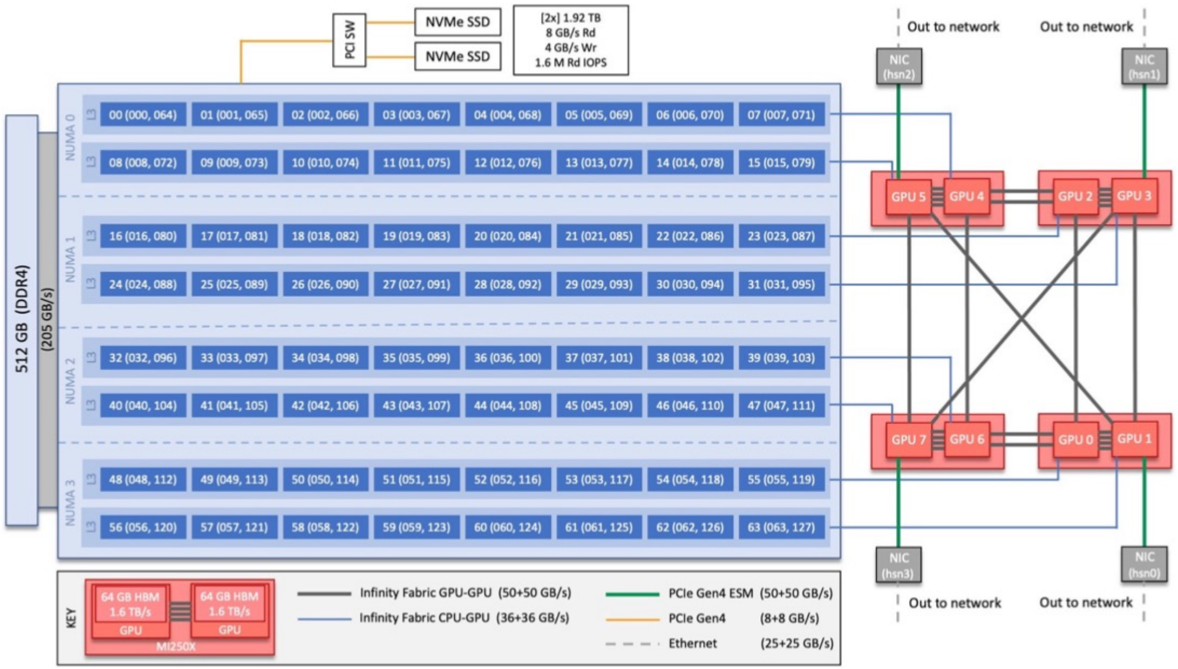
Message Size	Default	ALGO=Tree	ALGO=Ring
65536	0.85	0.85	0
131072	1.17	1.18	0
262144	1.41	1.42	0
524288	2.59	2.43	0.01
1048576	4.18	4.46	0.77
2097152	5.75	8.6	0.67
4194304	8.65	7.67	1.32
8388608	12.15	10	1.24
16777216	17.59	13.14	1.17
33554432	16.28	20.14	2.16
67108864	31.34	29.49	0.19
134217728	54.02	45.43	2.01
268435456	56.66	56.08	17.78
536870912	68.37	68.99	30.09
1073741824	70.81	70.77	57.92
2147483648	90.02	73.43	90.3
4294967296	90.31	75	90.45

CPU-GPU Binding

- Goal: match CPU cores to the GCD in the same L3 region
- RCCL self-checks the *GPU* and NIC topology
- Result: it doesn't need to bind to CPU cores ***if*** there's not much CPU-GPU transfer

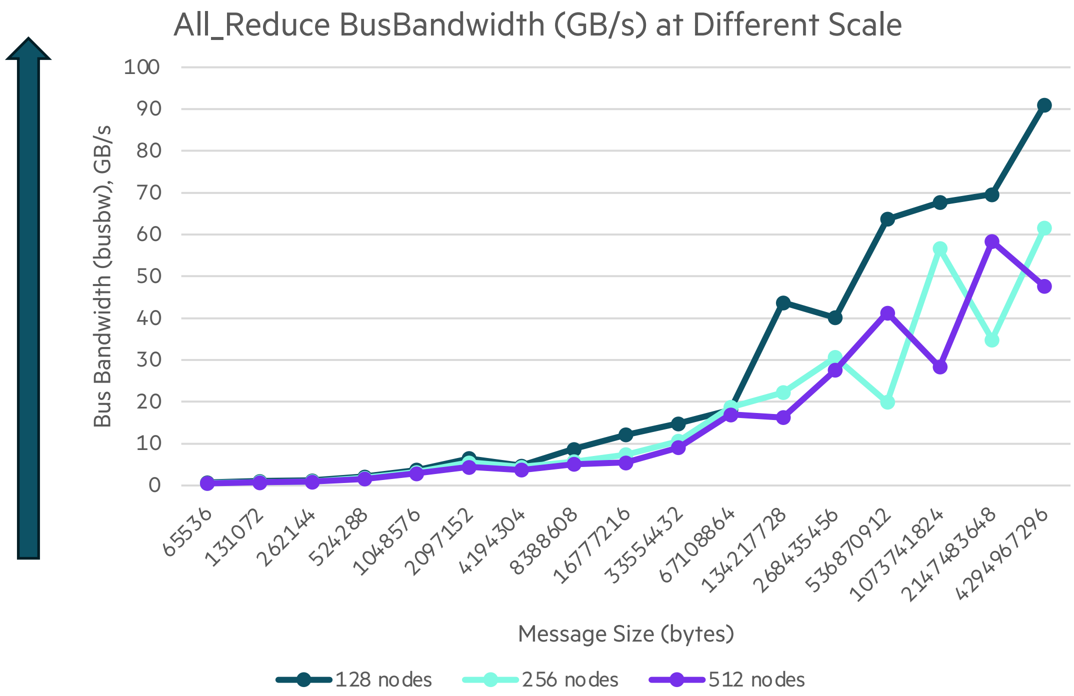
- Nevertheless, it won't hurt to bind, so use
 --gpus-per-task=1 --gpu-bind=closest
and *not*
--tasks-per-node=8

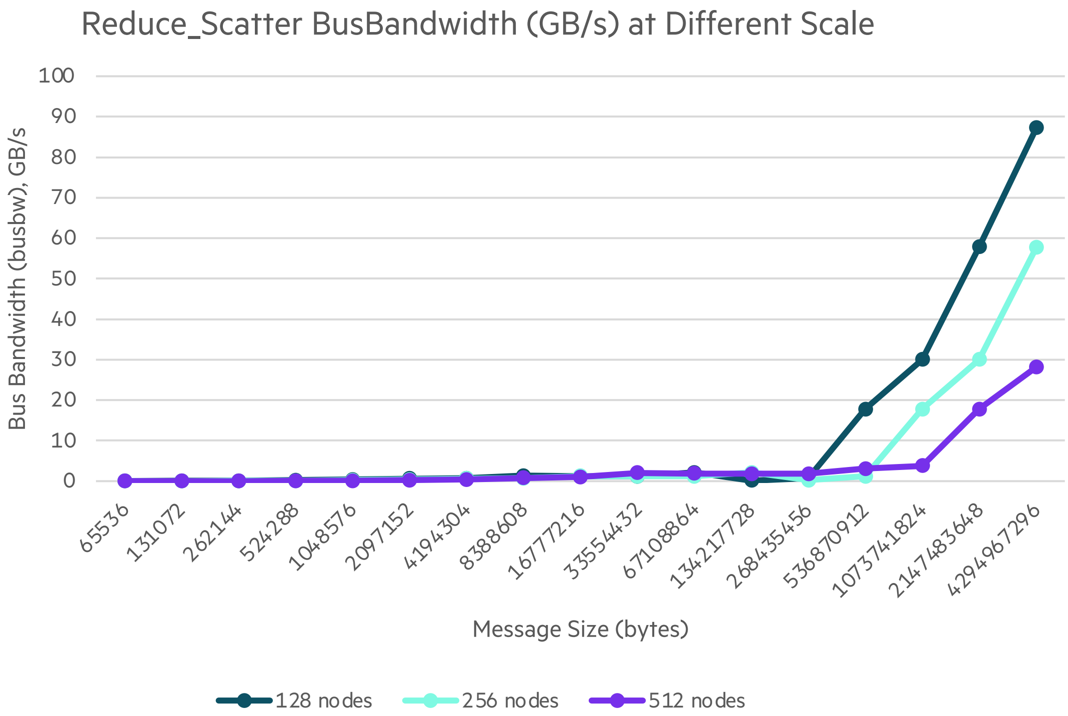
Frontier Node Architecture



Known Issues

- Does not perform well at larger scale (>= 256 nodes).





- Hang or crash when using more than 256 nodes and with NCCL_NET_GDR_LEVEL=3 set.
- Current plugin version (based on aws-ofi-nccl 1.4.0) is behind the latest aws-of-nccl version (1.8.1)

Part II: PyTorch Examples

A Harmless Message

- `[W socket.cpp:697] [c10d] The client socket cannot be initialized to connect to [crusher002.crusher.olcf.ornl.gov]:3442 (errno: 97 - Address family not supported by protocol).`

- After PyTorch v1.x, when using tcp to initialize PyTorch DDP, the default is to use IPv6 address; PyTorch falls back to use IPv4 if IPv6 does not work.

- The message is harmless, and it does not affect PyTorch-DDP uses RCCL as backend.

PyTorch Examples from AI-AT-SCALE-PART 2

- https://github.com/olcf/ai-training-series
- The training provides is a set of well-organized PyTorch-DDP examples. We use two of them in this talk:
  - launch_bigbird.crusher
  - launch_bigbird_srun.frontier
  - **launch_gptJ_srun.frontier**
  - **launch_bigbird.frontier**
  - launch_gptJ_fsdp.frontier
  - launch_gpt_srun.frontier

- Follow the instruction to prepare and download data:
  - git clone --recursive https://github.com/olcf/ai-training-series
  - In **ai_at_scale_part_2/ddp_examples** directory:

```
-bash download_oscar.sh
-bash dl_models.sh
```

Minor Modifications for Running the PyTorch Scripts

- Account
  #SBATCH -A **${your project account}**

- Environment
  ```
  source /lustre/orion/world-shared/stf218/sajal/miniconda3/bin/activate
  conda activate /lustre/orion/world-shared/stf218/sajal/TORCH2/env-py310-rccl
  ```
  ➔ This environment uses rocm/5.4.0 and pytorch-rocm/2.1.0, deepspeed/0.9.1.

- Add path to the plugin and the GDR variable
  ```
  #export NCCL_DEBUG=INFO
  export LD_LIBRARY_PATH=${PATH TO THE PLUGIN}/libs/
  #export FI_LOG_LEVEL=info
  export NCCL_NET_GDR_LEVEL=3
  ```

- Or use Sajal’s pre-build library
  ```
  export LD_LIBRARY_PATH=/lustre/orion/world-shared/stf218/sajal/aws-ofi-rccl/lib/:$LD_LIBRARY_PATH
  ```

Experiment Data - GPT-J

- **“sbatch launch_gptJ_srun.frontier”**
- Experiment setting: 16 nodes, 8 GPUs per node, max_steps=10, per_device_train_batch_size=4

- Without the plugin

  {'train_runtime': 1126.1257, **'train_samples_per_second': 36.372**, 'train_steps_per_second': 0.009, 'train_loss': 80.29856567382812, 'epoch': 0.09}

- With the plugin

  {'train_runtime': 671.7561, **'train_samples_per_second': 60.975**, 'train_steps_per_second': 0.015, 'train_loss': 79.96430053710938, 'epoch': 0.09}

- With the plugin + NCCL_NET_GDR_LEVEL=3

  {'train_runtime': 568.0539, **'train_samples_per_second': 72.106**, 'train_steps_per_second': 0.018, 'train_loss': 79.44228515625, 'epoch': 0.09}

Good performance improvement by using the plugin.

Experiment Data - BigBird Oscar

- **“sbatch launch_bigbird_srun.frontier”**
- Experiment setting: 10 nodes, 8 GPUs per node, 100 steps, per_device_train_batch_size =24
  - ”train_micro_batch_size_per_gpu” in ds_config.json is 4; change it to 24 to match per_device_train_batch_size.

- <u>Without the plugin</u>

  {'train_runtime': 208.6629, **'train_samples_per_second': 920.145**, 'train_steps_per_second': 0.479, 'train_loss': 6.68152099609375, 'epoch': 0.16}

- <u>With the plugin</u>

  {'train_runtime': 204.6703, **'train_samples_per_second': 938.094**, 'train_steps_per_second': 0.489, 'train_loss': 6.672891845703125, 'epoch': 0.16}

- <u>With the plugin + NCCL_NET_GDR_LEVEL=3</u>

  {'train_runtime': 203.6554, **'train_samples_per_second': 942.769**, 'train_steps_per_second': 0.491, 'train_loss': 6.687650756835938, 'epoch': 0.16}

Almost no performance improvement by using the plugin.

PyTorch Profiler Basic

- https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

```
with torch.profiler.profile(activities=[torch.profiler.ProfilerActivity.CPU,
torch.profiler.ProfilerActivity.CUDA]) ) as profiler:

    for epoch in range(epochs_trained, num_train_epochs):
        .
        training codes
        .
        .

print(profiler.key_averages().table(sort_by="cuda_time_total", row_limit=20))
print(profiler.key_averages().table(sort_by="cpu_time_total", row_limit=20))
```

PyTorch Profiler With Tracing

```
with torch.profiler.profile(
            activities=[torch.profiler.ProfilerActivity.CPU,
torch.profiler.ProfilerActivity.CUDA]),
            schedule=torch.profiler.schedule(
                wait=1,
                warmup=1,
                active=2,
                repeat=1),
            on_trace_ready=torch.profiler.tensorboard_trace_handler('./logs/gpt'),
 ) as profiler:

    for epoch in range(epochs_trained, num_train_epochs):
       .
       training codes
       .
       .
       profiler.step()


print(profiler.key_averages().table(sort_by="cuda_time_total", row_limit=20))
print(profiler.key_averages().table(sort_by="cpu_time_total", row_limit=20))
```

Modifications for Using PyTorch Profiler

- The BigBird Oscar script (bigbird_oscar_srun.py) uses custom_trainer.py, which has PyTorch Profiler enabled but does not print the performance data table.
  - Add the following two lines after the training loop (line 446):

```
print(profiler.key_averages().table(sort_by="cuda_time_total", row_limit=20))
print(profiler.key_averages().table(sort_by="cpu_time_total", row_limit=20))
```

- The GPT-J case uses the trainer provided by HuggingFace's Transformers; adding PyTorch Profiler codes requires more work.
- To keep things simple – modify the gptJ_oscar_srun.py script to use customer_trainer.py instead. The results show similar performance characteristic to using HuggingFace's Transformers's trainer.
  - In the file gptJ_oscar_srun.py, change the line:

```
trainer = Trainer
```

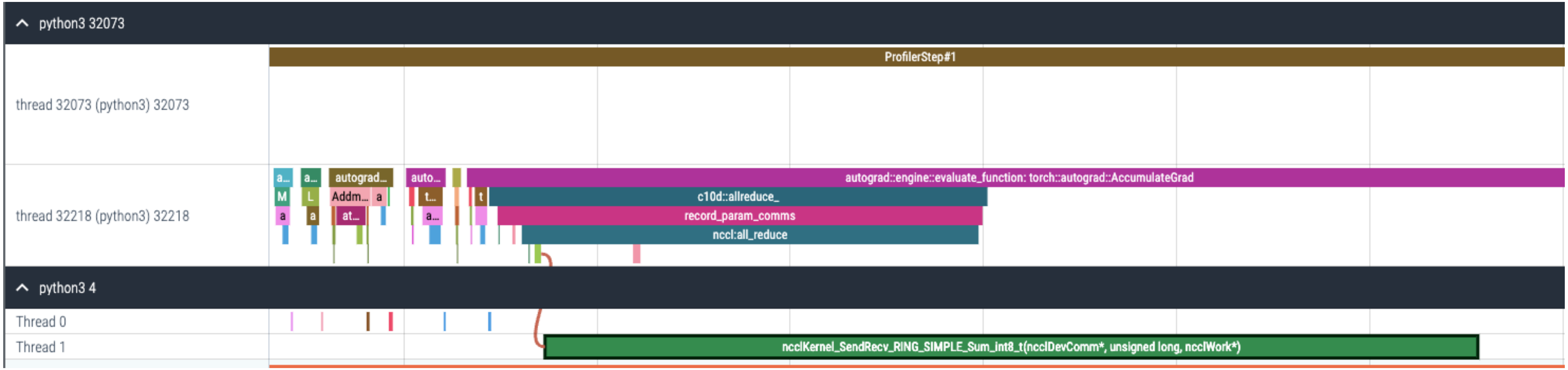
to

```
trainer = MyTrainer
```

Examining PyTorch Profiler Outputs

- There are three ways to view PyTorch Profiler outputs:
  - Text output
    - Basic PyTorch statistics shown at the end of a run.
  - Viewing tracing output use Perfetto
    - Need to download tracing files to local systems and then use tools such as Perfetto (https://ui.perfetto.dev/) to view the files.
  - Tensorboard-tb-plugin
    - Need to start a Tensorboard server on a Frontier login node, make an ssh connection, and then use a browser to connect to the login to view the data.
- PyTorch Profiler generates many types of performance data; the focus here is to evaluate information related to communications from the three types of output.

c10d Call Stack from a PyTorch Tracing File



- c10d is PyTorch's collective communication layer. When RCCL is used as the backend, PyTorch c10d uses RCCL collectives.
- This shows the call stack of c10d allreduce:

  c10d::allreduce ➔record_param_comms➔nccl:all_reduce➔ncclKernel_SendRecv_RING_SIMPLE_Sum_Int8_t().

GPT-J PyTorch Profiler Test Output

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	**CUDA total**	CUDA time avg	# of Calls
record_param_comms	0.22%	32.264ms	0.39%	56.688ms	41.078us	6478.273s	99.97%	7400.400s	5.363s	1380
Marker	0.00%	0.000us	0.00%	0.000us	0.000us	6465.912s	99.78%	6465.912s	3.156s	2049
autograd::engine::evaluate_function: torch::autograd...	0.30%	43.386ms	0.61%	88.287ms	154.889us	0.000us	0.00%	6401.640s	11.231s	570
c10d::reduce_	**0.02%**	**2.381ms**	**0.20%**	**29.019ms**	**43.835us**	**0.000us**	**0.00%**	**4681.067s**	**7.071s**	**662**
ProfilerStep*	1.53%	219.992ms	98.67%	14.228s	7.114s	0.000us	0.00%	2019.436s	1009.718s	2
hipStreamWaitEvent	0.03%	4.023ms	0.03%	4.814ms	2.326us	289.660s	4.47%	1261.031s	609.194ms	2070
hipLaunchKernel	0.19%	27.003ms	0.19%	27.003ms	3.919us	883.350s	13.63%	883.350s	128.208ms	6890
aten::copy_	0.11%	16.563ms	3.17%	457.239ms	71.713us	266.076ms	0.00%	745.987s	116.999ms	6376
hipMemcpyAsync	0.12%	17.476ms	0.12%	17.480ms	3.497us	587.764s	9.07%	587.764s	117.600ms	4998
c10d::allgather_	**0.01%**	**2.153ms**	**0.21%**	**30.103ms**	**1.254ms**	**0.000us**	**0.00%**	**391.278s**	**16.303s**	**24**
autograd::engine::evaluate_function: SliceBackward0	0.01%	820.000us	0.27%	38.749ms	21.575us	0.000us	0.00%	381.862s	212.618ms	1796
aten::slice_backward	0.03%	3.671ms	0.22%	31.241ms	17.395us	0.000us	0.00%	353.741s	196.960ms	1796
hipEventDestroy	0.00%	490.000us	0.00%	490.000us	0.357us	351.319s	5.42%	351.319s	256.063ms	1372
SliceBackward0	0.03%	5.026ms	0.23%	32.627ms	18.166us	0.000us	0.00%	317.616s	176.846ms	1796
hipExtModuleLaunchKernel	0.05%	7.576ms	0.05%	7.576ms	5.612us	210.276s	3.24%	210.276s	155.760ms	1350
hipMemcpyWithStream	91.35%	13.172s	91.35%	13.173s	92.765ms	40.958s	0.63%	180.501s	1.271s	142
hipStreamIsCapturing	0.00%	116.000us	0.00%	116.000us	0.168us	177.253s	2.74%	177.253s	256.889ms	690
aten::fill_	0.03%	3.903ms	0.07%	9.789ms	5.269us	9.936ms	0.00%	170.997s	92.033ms	1858
aten::item	-0.00%	-233.000us	88.44%	12.753s	579.668ms	0.000us	0.00%	160.403s	7.291s	22
aten::_local_scalar_dense	0.00%	520.000us	88.44%	12.753s	579.664ms	6.937ms	0.00%	160.403s	7.291s	22

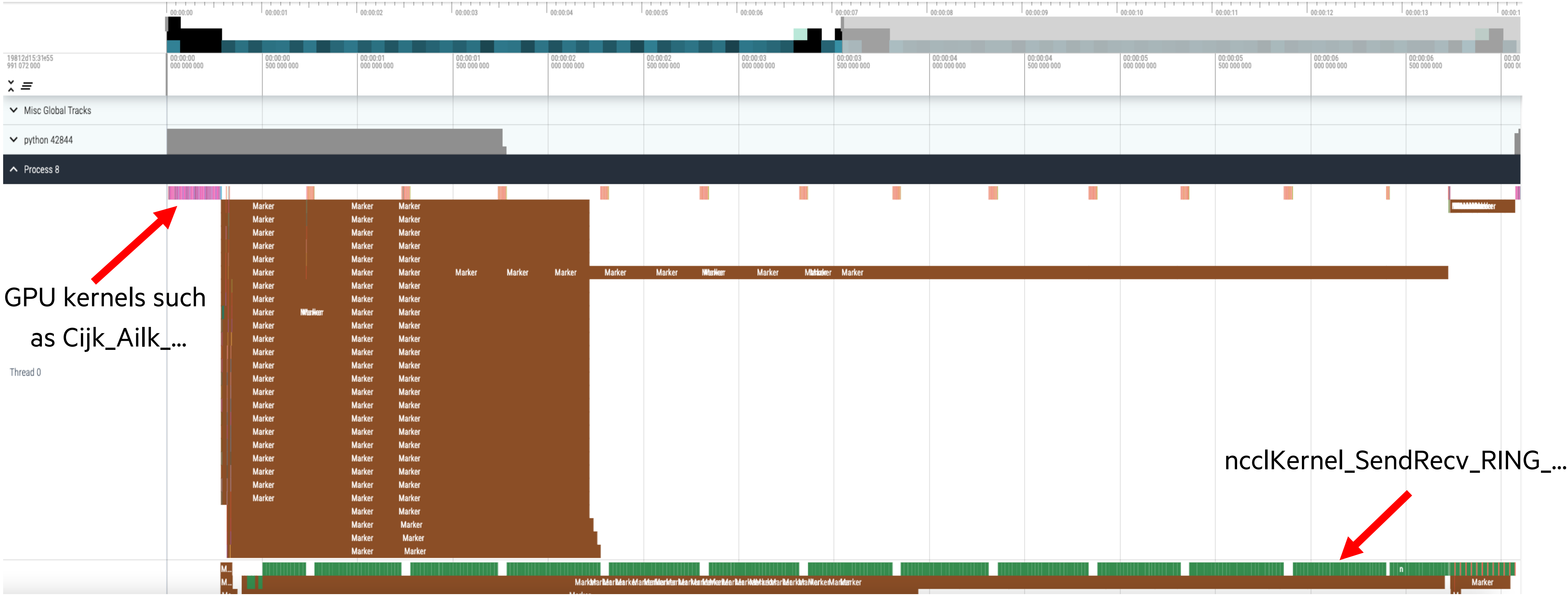
- A table generated by PyTorch Profiler can have (too) many columns. This is the minimum number of columns when setting to profile with ProfilerActivity.CPU and ProfilerActivity.CUDA.
- From the profiling output, c10d::reduce time is significant.

BigBird Oscar PyTorch Profiler Test Output

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg
autograd::engine::evaluate_function: SliceBackward0	0.06%	3.862ms	1.19%	80.099ms	32.403us	0.000us	0.00%	1.425s	576.324us
ProfilerStep*	44.69%	3.007s	57.13%	3.845s	1.922s	0.000us	0.00%	1.403s	701.365ms
aten::slice_backward	0.13%	8.933ms	0.89%	59.797ms	24.190us	0.000us	0.00%	1.117s	451.718us
SliceBackward0	0.15%	10.059ms	0.95%	64.199ms	25.970us	0.000us	0.00%	1.032s	417.547us
aten::copy_	0.20%	13.452ms	18.30%	1.232s	265.930us	765.246ms	19.32%	944.254ms	203.854us
autograd::engine::evaluate_function: AddmmBackward0	0.02%	1.599ms	0.23%	15.242ms	102.986us	0.000us	0.00%	816.697ms	5.518ms
AddmmBackward0	0.03%	1.893ms	0.15%	9.883ms	66.777us	0.000us	0.00%	761.703ms	5.147ms
aten::mm	0.07%	4.745ms	0.10%	6.544ms	22.108us	746.361ms	18.85%	761.703ms	2.573ms
CopyDeviceToDevice	0.00%	0.000us	0.00%	0.000us	0.000us	586.327ms	14.81%	586.327ms	210.606us
aten::fill_	0.13%	8.684ms	0.34%	22.976ms	6.896us	361.707ms	9.13%	494.488ms	148.406us
hipLaunchKernel	0.68%	45.624ms	0.68%	45.624ms	4.351us	471.582ms	11.91%	471.582ms	44.973us
aten::zero_	0.07%	4.596ms	0.39%	25.986ms	8.050us	0.000us	0.00%	467.822ms	144.926us
aten::add_	0.10%	6.835ms	0.18%	11.806ms	8.433us	377.011ms	9.52%	437.015ms	312.154us
aten::zeros	0.13%	9.015ms	0.58%	39.004ms	12.367us	0.000us	0.00%	426.669ms	135.279us
aten::bmm	0.29%	19.395ms	7.86%	529.127ms	459.312us	339.261ms	8.57%	385.911ms	334.992us
void at::native::modern::elementwise_kernel<at::nati...	0.00%	0.000us	0.00%	0.000us	0.000us	380.209ms	9.60%	380.209ms	271.190us
aten::linear	0.02%	1.310ms	0.13%	8.787ms	58.580us	0.000us	0.00%	369.570ms	2.464ms
aten::addmm	0.06%	4.281ms	0.09%	6.102ms	40.680us	359.142ms	9.07%	369.570ms	2.464ms
void at::native::modern::elementwise_kernel<at::nati...	0.00%	0.000us	0.00%	0.000us	0.000us	360.402ms	9.10%	360.402ms	109.946us
aten::mul	0.13%	8.472ms	0.19%	13.113ms	11.878us	257.161ms	6.49%	316.466ms	286.654us

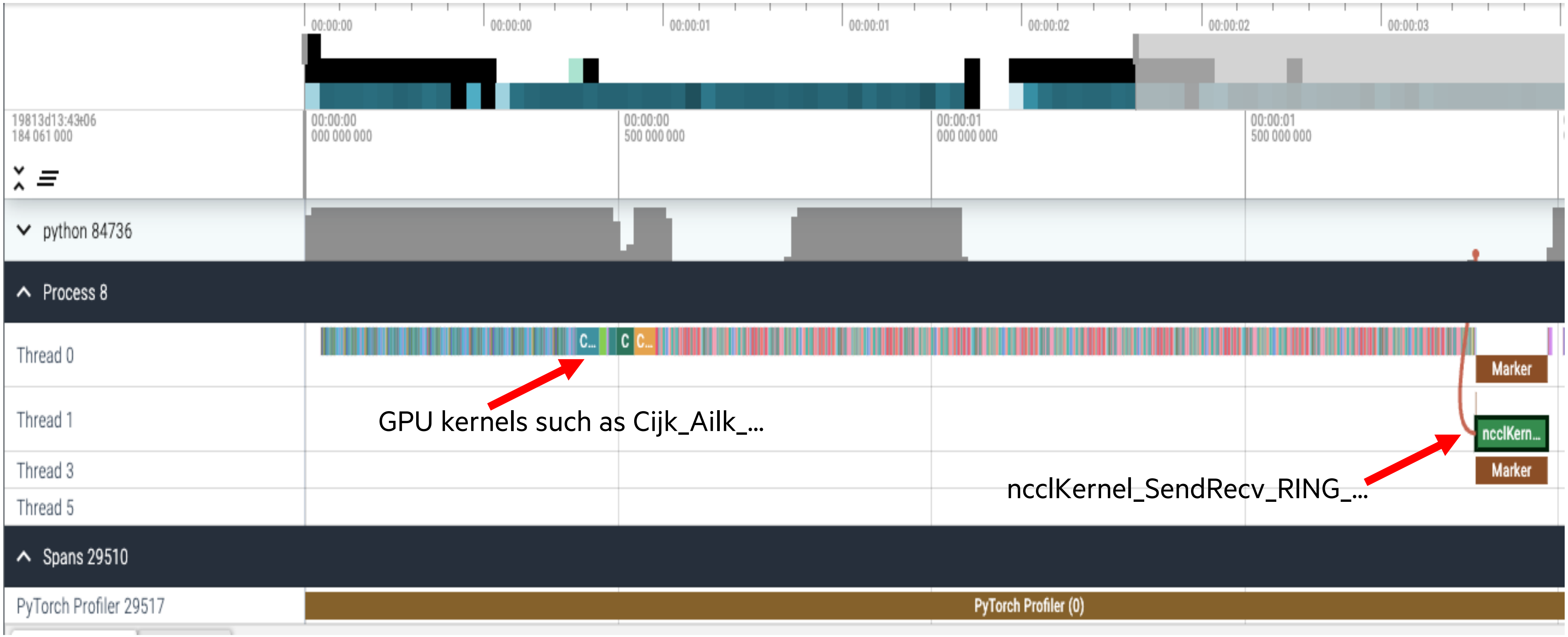
- c10d time is nowhere to be found in the profiling output.

PyTorch Profiler Tracing: GPT-J



- This is the view of one epoch (one step).
- All the green blocks are ncclKernel_SendRecv_RING_SIMPLE calls.

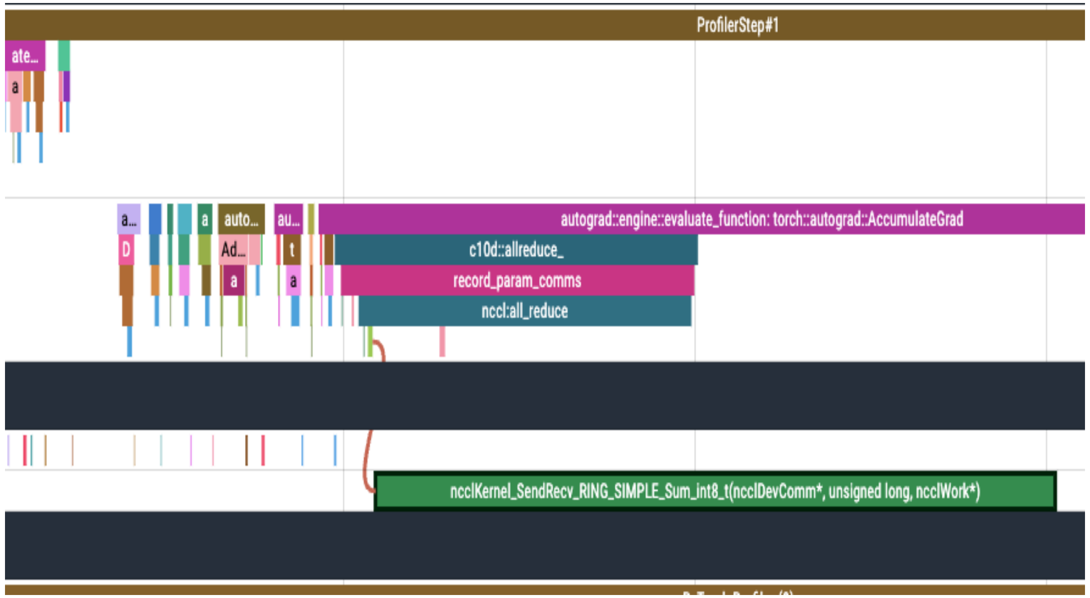
PyTorch Profiler Tracing – BigBird Oscar



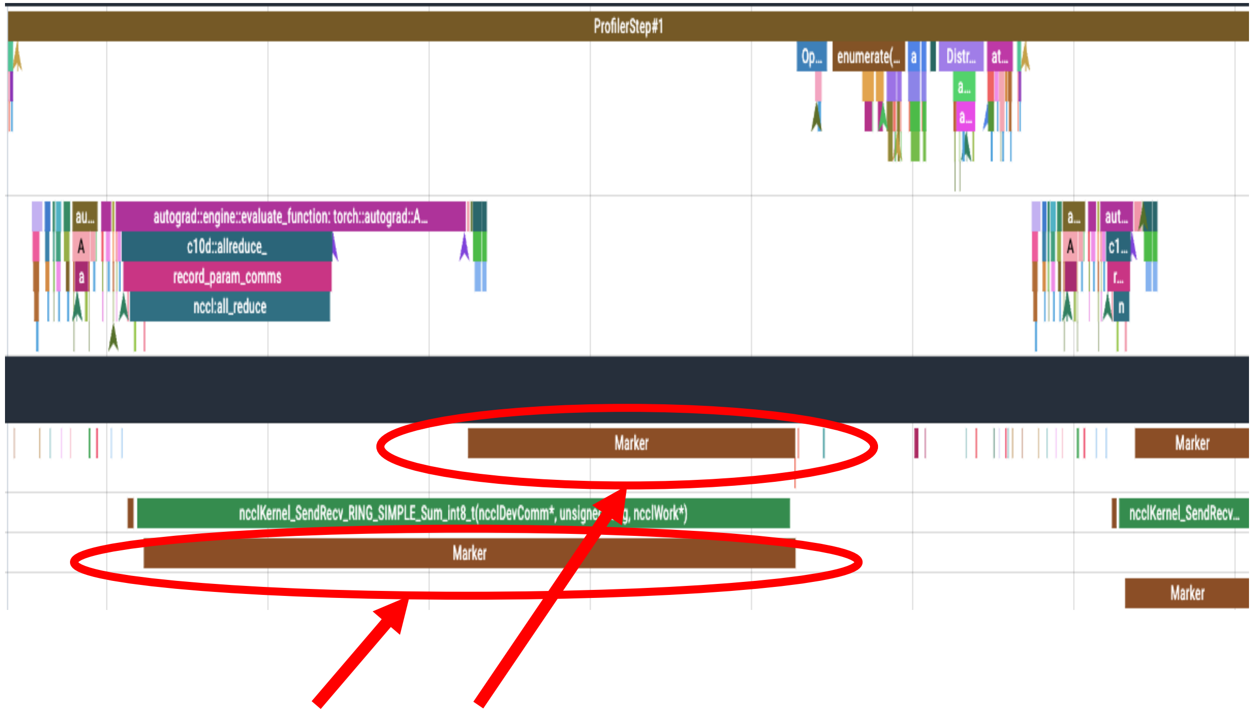
- This is the view of one epoch (one step).
- The BigBird Oscar case only has one ncclKernel_SendRecv_RING_SIMPLE call.

The “Marker” Issue in PyTorch-rocm Tracing Files

PyTorch-rocm/2.1.2



PyTorch-rocm/2.0.1



- Pytorch-rocm prior to 2.1.2 generate many “Marker” blocks in tracing files.
- PyTorch-rocm/2.1.2 and later releases have the issue fixed.

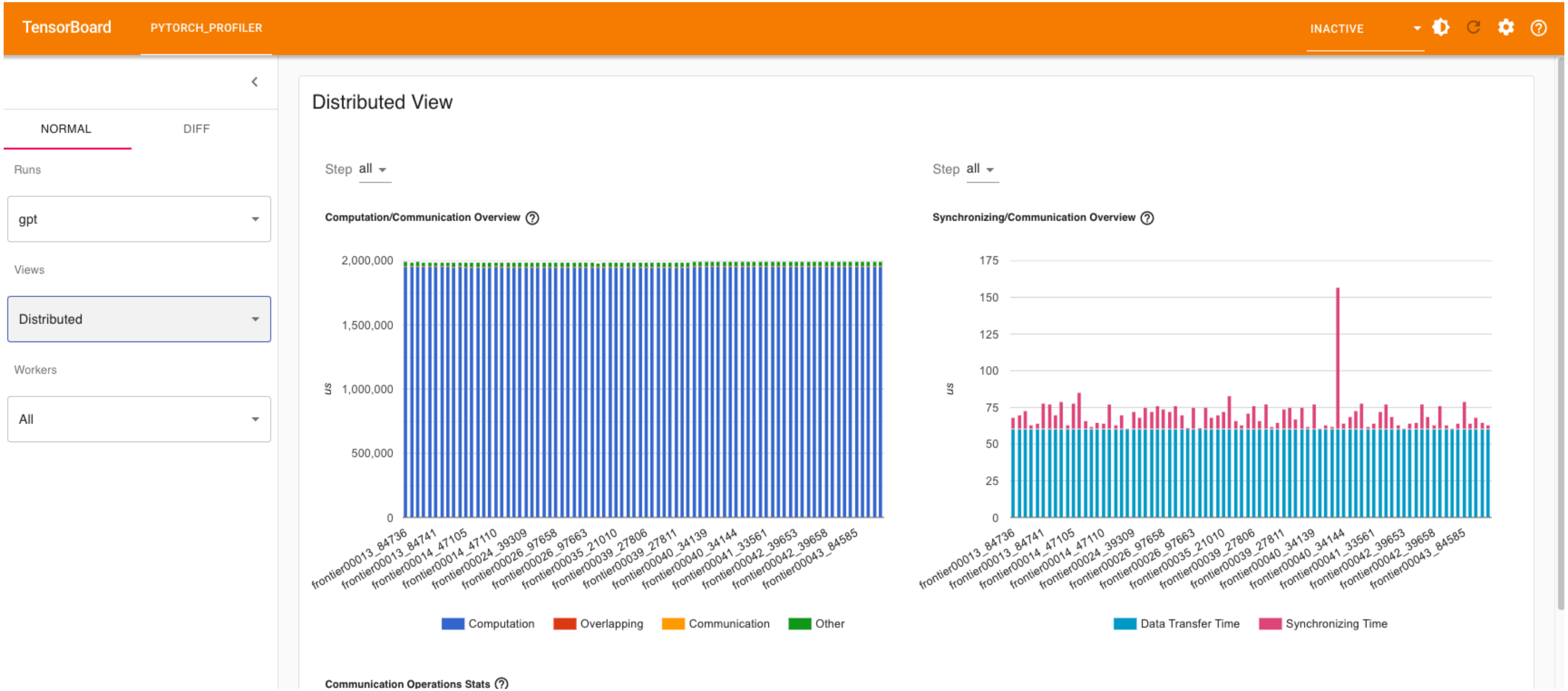
PyTorch Profiler with Tensorboard

- https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html

1. Install the Tensorboard Plugin package with
   pip install torch_tb_profiler
2. Start tensorboard on a Frontier login node with
   tensorboard --logdir ${PATH_TO_LOG_DIRECTORY} --host localhost
3. ssh from local system to the Frontier node, for example:
   ssh -L 6006:localhost:6006 $USER@login04.frontier.olcf.ornl.gov
4. Open browser at **http://localhost:6006** should work.

- The "Distributed View" is supposed to provide information about communication vs. computation. However, this view does not always show up.

Tensorboard PyTorch Plugin – BigBird Oscar



- The Distributed View is not available from tracing output of the GPT-J example.

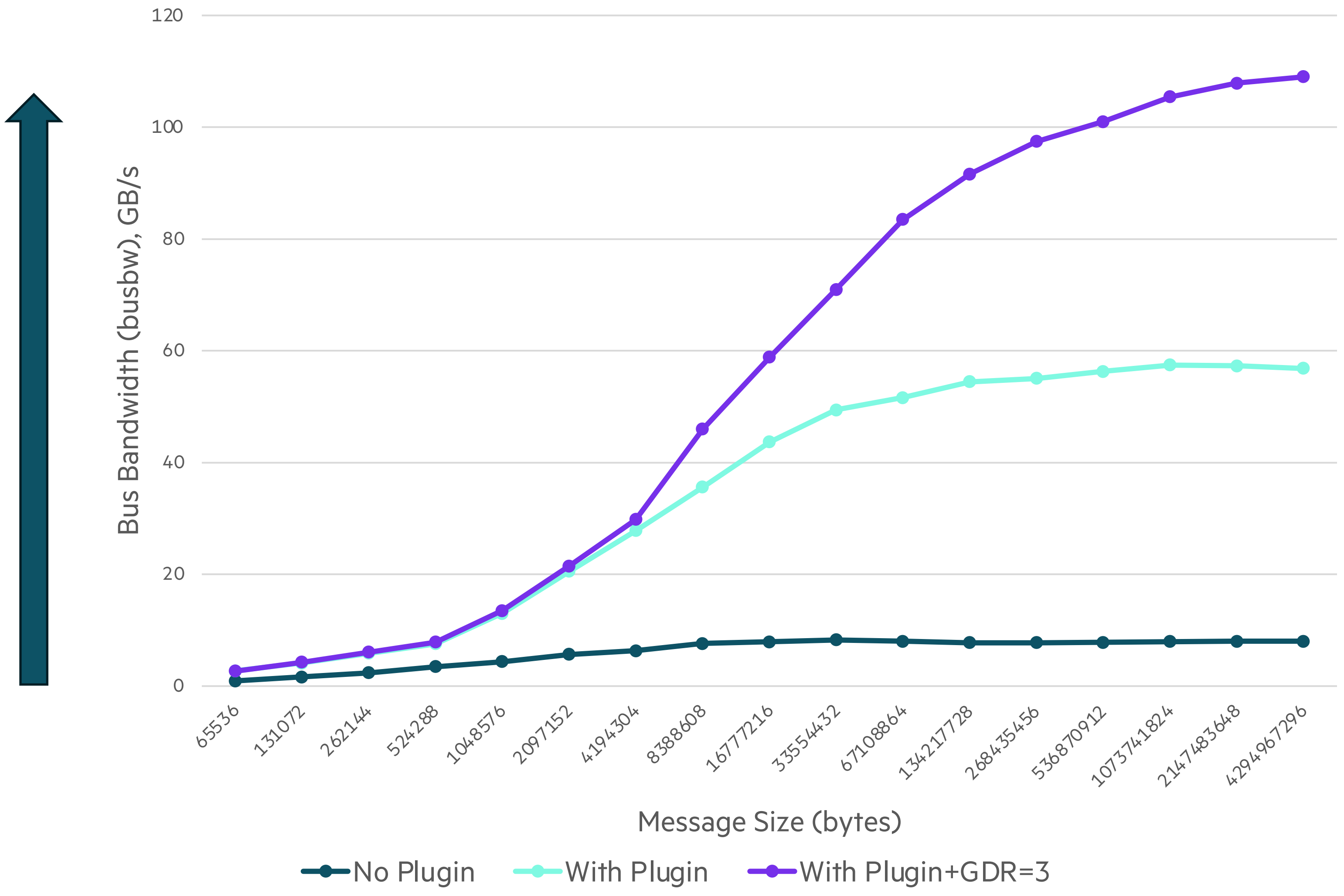
PyTorch Profiler and the Communication Cost

- Both profiling text output and tracing output show the GPT-J case is communication intensive while BigBird Oscar case is computationally intensive.
  ➔The plugin improves communication time and thus only the GPT-J case can benefit from using the plugin.

- The TensorBoard PyTorch plugin could be useful in the future if the Distributed View could be available for every DDP run.

- PyTorch Profiler does not provide statistics of communication such as how many times a certain message size was used during a training run.

Revisit RCCL All_reduce on 2 nodes, 16 ranks (-n16 -N2)



Message Size	No Plugin	With Plugin	With Plugin+GDR=3
65536	0.93	2.74	2.67
131072	1.62	4.13	4.3
262144	2.38	5.85	6.07
524288	3.48	7.58	7.85
1048576	4.4	13.02	13.52
2097152	5.67	20.52	21.48
4194304	6.33	27.8	29.83
8388608	7.6	35.58	45.99
16777216	7.9	43.7	58.85
33554432	8.25	49.4	70.98
67108864	8.04	51.63	83.46
134217728	7.79	54.43	91.62
268435456	7.78	55.03	97.46
536870912	7.82	56.29	101
1073741824	7.96	57.43	105.42
2147483648	8	57.32	107.88
4294967296	8.03	56.85	109.01

- Small message sizes = less performance improvement.
- Message sizes < 4MB, GDR=3 does not help much.

New HPE Perftools NCCL/RCCL Profiling Feature

- The steps to use Perftools to profile Python codes:
  - ”module load perftools-base perftools-preload”
  - Instead of “srun … python ${your_script}”:
    - **export PAT_RT_TRACE_PYTHON_GROUPS="pytorch**"
    - srun … **pat_run -w –g rccl** python ${your_script}

Table 1: Time and Bytes Transferred for Accelerator Regions

	Time%	Time	Acc Time%	Acc Time	Acc Copy In (MiBytes)	Acc Copy Out (MiBytes)	Acc To Acc (MiBytes)	Events	Function Source Line Accelerator ID=HIDE PE=HIDE Thread=HIDE
	100.0%	601.980657	100.0%	361.69	854.88	0.01	53.14	1,780,200	Total
	73.9%	444.819407	0.0%	0.00	4.57	0.00	0	4,689	main
Backward()	14.9%	89.825998	--	--	--	--	0	0	python.Tensor.backward
									python3.10/site-packages/torch/_tensor.py
	3								line.428
	6.9%	41.518919	70.7%	255.79	0.00	0.01	0	1,298,951	python.train
									distributed/FSDP/utils/train_utils.py
	3 6.9%	41.517834	70.5%	254.97	0.00	0.01	0	1,289,915	line.56
Forward()	1.9%	11.511038	--	--	--	--	0	0	python.FullyShardedDataParallel.forward
									torch/distributed/fsdp/fully_sharded_data_parallel.py
	3								line.727
	1.2%	7.117145	10.8%	39.16	--	--	0	213,526	python.validation
									distributed/FSDP/utils/train_utils.py
	3 1.2%	7.116681	10.6%	38.32	--	--	0	213,438	line.96
Allgather()	0.3%	2.017215	17.8%	64.26	--	--	0	14,657	python.all_gather_into_tensor
									site-packages/torch/distributed/distributed_c10d.py
	3								line.2532
	0.2%	1.352622	0.0%	0.07	756.18	--	53.14	1,258	python.fsdp_main
									examples/distributed/FSDP/T5_training.py
	3 0.2%	1.326641	0.0%	0.04	756.18	--	53.14	1,181	line.148
	0.0%	0.009213	0.0%	0.00	94.13	--	0	15	python.to
									python3.10/site-packages/transformers/modeling_utils.py
	3								line.2576

A Warning About Using Profiler or Enabling Debugging Messages

- Enabling debugging variables, such as NCCL_DEBUG or FI_LOG_LEVEL, can generate many messages and create large files.
- When using PyTorch Profiler with tracing on, each rank may generate a large file; at large scale the outputs from PyTorch Profiler may use a large amount of disk space.
- The overhead of using a profiler sometimes can be significant and can distort the collected performance data – always double check with the original performance data.

Some Environment Variables for the RCCL Layer

- On the PyTorch layer
  - There is no environment variables for the c10d layer.
  - At the moment, setting NCCL as the c10d backend is the only option to run PyTorch at scale on Frontier.
  - If you're not seeing performance improvement with the plugin – try a profiler and evaluate communication cost.

- On the RCCL layer
  - NCCL_NET_GDR_LEVEL=3 - Remove this setting if you encounter a hang/crash.
  - NCCL_ALGO=TREE or RING – May see performance difference with this setting.
  - NCCL_CROSS_NIC=1 - On large systems, this NCCL setting has been found to improve performance.
  - NCCL_DEBUG=info for debugging.

Some Environment Variables for the Plugin Layer

- On the OFI-plugin layer
  - Libfabric/CXI variables are mostly for debugging or walkaround purpose.
    - **FI_LOG_LEVEL=warn FI_LOG_PROV=cxi** – print only CXI related messages.
    - **FI_MR_CACHE_MONITOR=userfaultfd** – The memory cache monitor is responsible for detecting system memory changes made between the virtual addresses used by an application and the underlying physical pages. Userfaultfd is a Linux kernel feature used to report virtual to physical address mapping changes to user space.
    - **FI_CXI_DISABLE_HOST_REGISTER=1** – Disable registration of host buffers (overflow and request) with GPU
    - **FI_CXI_DEFAULT_CQ_SIZE** – the default is 131072.
    - **FI_CXI_DEFAULT_TX_SIZE** – the default is 256.

- Encounter a hang or crash when using the plugin:
  - Frontier Office Hours.
  - File an OLCFHELP ticket. If possible, a reproducer would be very helpful.

Summary

- Topics we talked about:

  - In Part I:
    - Basic about RCCL, how to build and use the RCCL Tester to evaluate RCCL performance.
    - The OFI-Plugin, and how it improve performance of the RCCL Tester.

  - In Part II:
    - Examples showing performance with and without using the plugin.
    - How to use PyTorch Profiler, and different methods to examine performance data generated by PyTorch Profiler.

Thank you

Mengshiou Wu – meng-shiou.wu@hpe.com
Mark Stock – mark.stock@hpe.com

Message Histograms from Perftools

Table 1: Time and Bytes Transferred for Accelerator Regions

Time Data				Host-To-Accelerator Transfer Data							Accelerator-to-Host Transfer Data			Accelerator-to-Accelerator Transfer Data				
Time%	Time	Acc Time%	Acc Time	Acc Copy In (MiBytes)	HtoA CpySz <16 Count	16<= HtoA CpySz <256 Count	256<= HtoA CpySz <4KiB Count	4KiB<= HtoA CpySz <64KiB Count	1MiB<= HtoA CpySz <16MiB Count	16MiB<= HtoA CpySz Count	Acc Copy Out (MiBytes)	AtoH CpySz <16 Count	16<= AtoH CpySz <256 Count	Acc To Acc (MiBytes)	64KiB<= AtoA CpySz <1MiB Count	1MiB<= AtoA CpySz <16MiB Count	Events	Function Source Accelerator ID=HIDE PE=HIDE Thread=HIDE
100.0%	192.144116	100.0%	344.88	854.88	58.0	0.0	80.0	448.0	192.0	1.0	0.01	6,121.2	0.0	53.14	0.0	25.0	1,765,354	Total
20.3%	38.945723	70.3%	242.46	0.00	50.0	0.0	0.0	0.0	0.0	0.0	0.01	6,000.0	0.0	0	0.0	0.0	1,286,371	python.train distributed/FSDP/utils/train_utils.py
16.5%	31.724635	0.0%	0.00	4.57	8.0	0.0	16.0	448.0	0.0	0.0	0.00	121.2	0.0	0	0.0	0.0	4,732	main
3.6%	6.871981	11.2%	38.49	--	0.0	0.0	0.0	0.0	0.0	0.0	--	0.0	0.0	0	0.0	0.0	211,308	python.validation distributed/FSDP/utils/train_utils.py
1.3%	2.449936	0.0%	0.04	850.31	0.0	0.0	64.0	0.0	192.0	1.0	--	0.0	0.0	53.14	0.0	25.0	1,196	[R]Initialization phase User-defined Region examples/distributed/FSDP/my_fsdp.py
1.2%	2.273711	17.8%	61.46	--	0.0	0.0	0.0	0.0	0.0	0.0	--	0.0	0.0	0	0.0	0.0	14,665	python.all_gather_into_tensor site-packages/torch/distributed/distributed_c10d.py

TensorFlow Example

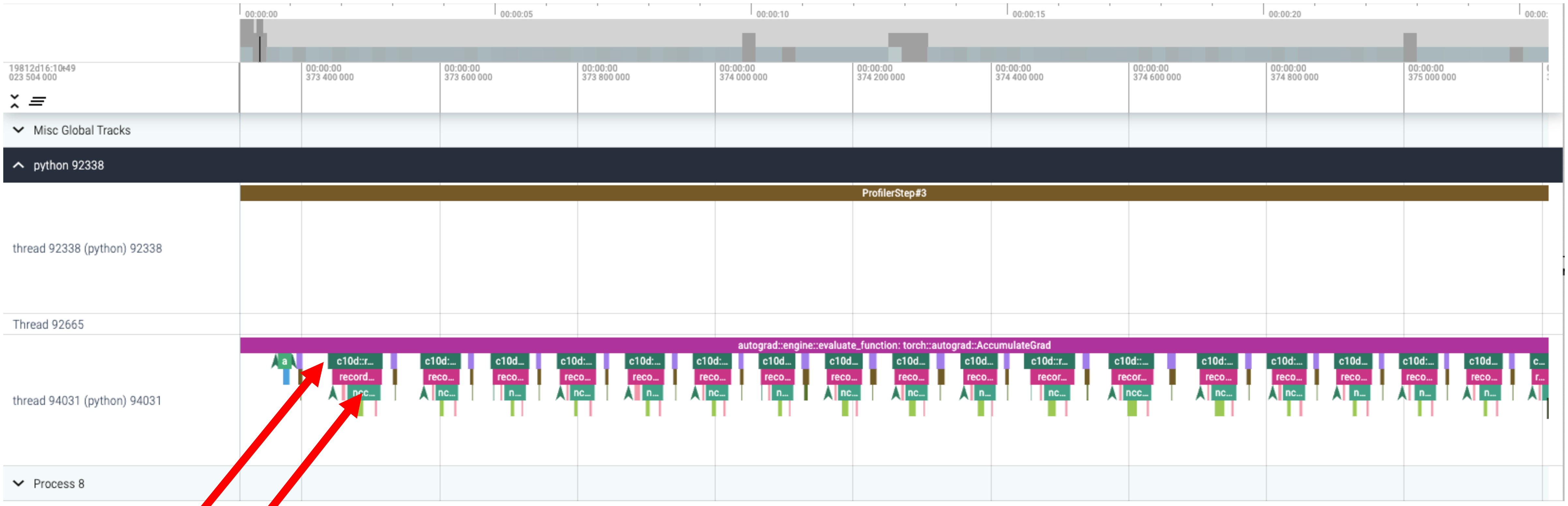
- Other MLDL frameworks or codes that use RCCL may benefit from using the plugin.

- TensorFlow tf_cnn_benchmark.py code:
  - Use TensorFlow 2.9 + Horovod/RCCL backend built with ROCm/5.6.0, batch size = 128

	No Plugin	With the Plugin	With the Plugin + GDR=3
16 nodes, 128 ranks	118874.19	129670.74	133778.7
32 nodes, 256 ranks	219602.73	252821.21	261120.75
64 nodes, 512 ranks	388282.87	477924.70	508689.71

– Numbers = Images per second, higher is better.

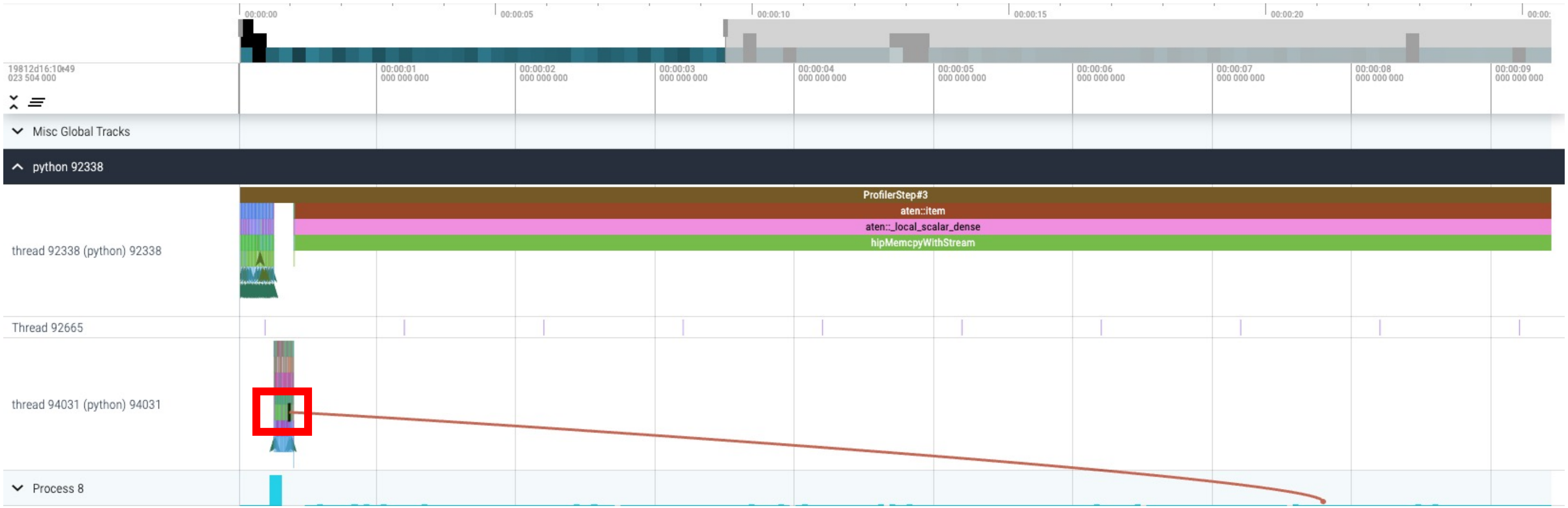
PyTorch Profiler Tracing: GPT-J



c10d::reduce nccl::reduce

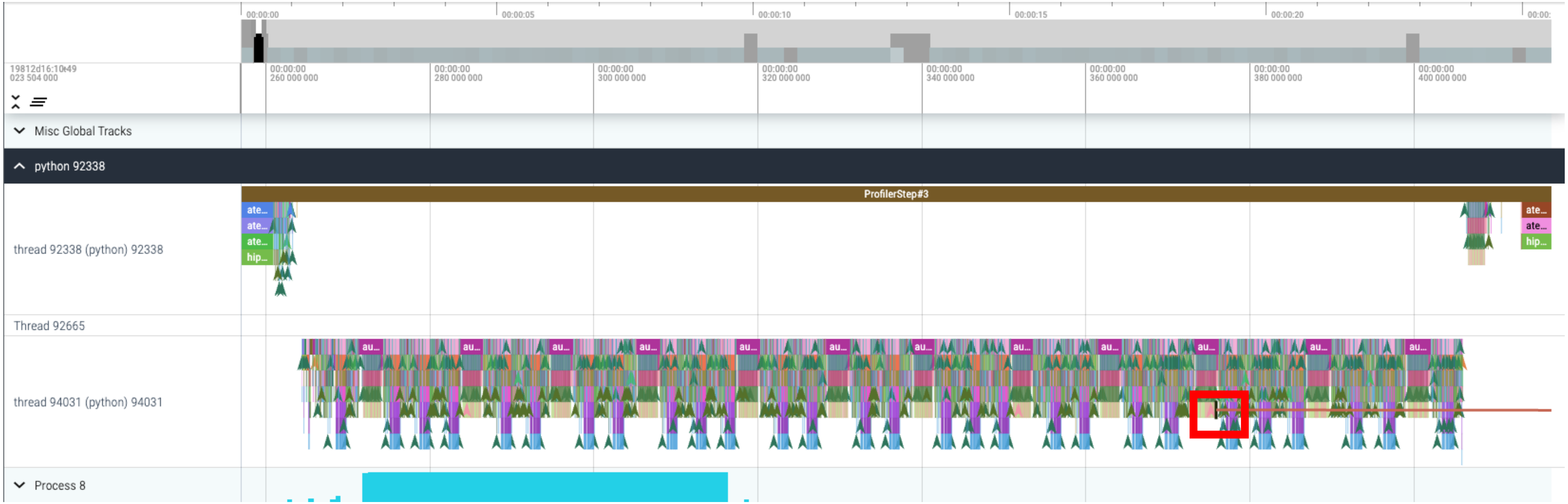
- A zoom in view to a torch::autograd::AcumulatGrad step.
- There are many c10d calls.

PyTorch Profiler Tracing: GPT-J



- Zoom out further. The line shows a c10d/nccl reduce call from an early stage that later invokes ncclKernel_SendRecv_RING_SIMPLE
- The red box is the viewing area in the previous slide.

PyTorch Profiler Tracing: GPT-J



- A zoom out view of the figure in the previous slide.
- The red box is the viewing area in the previous slide.