Tools in action

An example with Pytorch

Sam Antao

LUMI Advanced Training
Mar. 7th 2025

AMD
together we advance_

slides on LUMI in /project/project_465001726/Slides/AMD/

hands-on exercises: https://hackmd.io/@sfantao/lumi-training-sto-2025#Pytorch-example

hands-on source code: /project/project_465001726/Exercises/AMD/Pytorch

Agenda

1. **Intro to Pytorch and its dependencies**
2. **Controlling affinity**
3. **Profiling – rocprof and omnitools.**
4. **Debugging**

All suggestions presented here are transversal to any AI or HPC application!

Python applications can leverage the same tooling as non-python applications!

Scripting examples are suggestions and can always be adapted!

Pytorch highlight

- Official page: https://pytorch.org/
- Code: https://github.com/pytorch/pytorch
- Python™-based framework for machine learning
  - Auto-differentiation on tensor types
- GPU-enabled
  - ROCm support for MI250x (and others)
  - Hipification as part of the build system
    - C/C++ libraries with proper bindings for Python
  - Python code does **NOT** need changing – using the same CUDA conventions
- Other related packages:
  - Torch vision/audio, triton, many others
- Many more build on it
  - vLLM, Deepspeed, Megatron-LM

Pytorch install – our base environment

```
module purge
module load CrayEnv
module load PrgEnv-cray/8.5.0
module load craype-accel-amd-gfx90a
module load cray-python
```

Setup the GPU environment and the Cray Python environment

Recent Pytorch builds need recent user-level ROCm versions.

```
# This path provides more recent ROCm modules.
module use /appl/local/containers/test-modules
module load rocm/6.1.3.lua
```

We will be using 6.1.3 as it the oldest version for Pytorch 2.4.1 official releases

Pytorch install – running the examples

- For simplicity and improve the demonstration we leverage interactive runs on existing node allocation
- We run beforehand:

```
N=1 ; salloc -p standard-g \
        --threads-per-core 1 \
        --exclusive \
        -N $N \
        --gpus $((N*8)) \
        -t 4:00:00 --mem 0
```

We are reserving **N nodes**, in this case only one node, using **one of the two available hardware threads** per core. We we'll be using the **8 GCDs** available in each node.

- This is a good way to experiment and converge to the correct job description.
- Don't forget to release your allocations once you are done!
- Once you consolidate your job description you can leverage batch jobs.
  - Salloc options translate directly to sbatch options.

Pytorch install – system python

- Native install from Pytorch python wheels

Package install version can mix the Pytorch version as well as the ROCm it was build against.

This is where Pytorch project posts the wheel files - browse it to see what versions and ROCm combinations are available.

Where do we want to install things - don't use your home folder!

```
pip3 install -t $PWD/pip-installs --pre torch==2.4.1+rocm6.1      --index-url https://download.pytorch.org/whl/
```

```
Collecting torch==2.4.1+rocm6.1
  Downloading https://download.pytorch.org/whl/rocm6.1/torch-2.4.1%2Brocm6.1-cp311-cp311-linux_x86_64.whl (2662.0 MB)
                                                  0.9/2.7 GB 74.9 MB/s eta 0:00:24
```

Make the frershly install Pytorch available to your Python runs

```
PYTHONPATH=$PWD/pip-installs \
        srun --jobid=$jobid -n1 --gpus 8 \
        python -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```

> I have this many devices: 8

Should yield the number of GCDs in the node.

Example 01

Pytorch install – virtual environments

Example 02

- Virtual environments are convenient to manage python package installation in ones user-space

Leverage the venv module to create the virtual environment

We are happy to leverage system's already installed packages

The folder where the virtual environment will be installed

```
python -m venv --system-site-packages cray-python-virtualenv
```

```
source cray-python-virtualenv/bin/activate
```

Activate the environment. It will be leveraged by the install and run.

Install and run as before. No need to specify install location – the environment is doing it for you.

```
pip3 install --pre torch==2.4.1+rocm6.1 --index-url https://download.pytorch.org/whl/
srun --jobid=$jobid -n1 --gpus 8 \
 python -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```

Pytorch install – conda environment

Example 03

- Conda environment adds the package-manager functionality to a virtual environment
- One can tune the Python version to use as we won't be leveraging the system one anymore.
  - No `module load cray-python` needed!

```
curl -LO https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash ./Miniconda3-* -b -p miniconda3 -s
```

Download and install a minimal conda (miniconda).

```
source $PWD/miniconda3/bin/activate base
```

Activate the conda environment

```
conda create -y -n pytorch python=3.11
source $PWD/miniconda3/bin/activate pytorch
```

Create and activate a conda environment to install Pytorch based on Python 3.11

Install and run as before - Conda package manager doesn't have ROCm enabled Pytorch installs

```
pip3 install --pre torch==2.4.1+rocm6.1 --index-url https://download.pytorch.org/whl/
srun --jobid=$jobid -n1 --gpus 8 \
  python -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'
```


AMD together we advance_

Pytorch install – conda environment install from source

- Installing Pytorch from source is not recommended on LUMI
  - Too old default ROCm to build against.
- It might be useful in some cases: builds with symbols for debugging.

```
# We need a recent enough compiler – we'll use gcc
module load PrgEnv-gnu/8.5.0 gcc/10.3.0

# Clone a given version of Pytorch and all its third_party components

git clone -b v2.4.1 --recursive \
  https://github.com/pytorch/pytorch pytorch-source

# Createand activate conda environment to manage the install
conda create -y -n pytorch-from-source python=3.11
source $wd/miniconda3/bin/activate pytorch-from-source

# Install requirements for Pytorch and some build tools.
pip install -r $wd/pytorch-source/requirements.txt
conda install -y cmake ninja

# Sometimes we need to solve some library clashes between conda and the system. We force the
removal of the conda libstdc++ so that we use the system one.
rm -rf $wd/miniconda3/envs/pytorch-from-source/lib/libstdc++.so
```

```
# Point to our ROCM instalation that is not in a default path.
grep -rl /opt/rocm | \
  xargs sed  -I "s#/opt/rocm#$ROCM_PATH#g"

# Hipify source
nice python3 tools/amd_build/build_amd.py

# Build with debug symbols
CC=$(which gcc) \
  CXX=$(which g++) \
  CMAKE_PREFIX_PATH=$CONDA_PREFIX:$CMAKE_PREFIX_PATH \
  LDFLAGS="-L$ROCM_PATH/deps -lstdc++ -ltinfo" \
  USE_KINETO=0 BUILD_TEST=0 \
  PYTORCH_ROCM_ARCH=gfx90a \
  REL_WITH_DEB_INFO=1 \
  nice python3 setup.py bdist_wheel

pip install $wd/pytorch-source/dist/torch-*.whl
```

Enable/disable the packages you care about.

Build with debug symbols

Pytorch install – Singularity containers

- Control better the Pytorch environment
- Less strain on the filesystem
  - All application installation is loaded as a single file
- Enable more recent ROCm versions
- Transferable and arguably more portable
- Some containers available under:
  - /appl/local/containers/sif-images/

- Any cons?
  - Updating the environment and installing more packages may require rebuild the container
  - Containers can't currently be build on LUMI:
    - Needs containers to be built elsewhere and copied to the system
  - Submitting jobs has to be done more carefully.

```
SIF=<myimage.sif>

srun --jobid=$jobid -n1 \
singularity exec \
    -B /var/spool/slurmd \
    -B /opt/cray \
    -B /usr/lib64/libcxi.so.1 \
    -B $wd:/workdir \
    $SIF /workdir/run-me.sh
```

Make relevant pieces of native environment visible inside the container

Make my work directory visible inside the container

The container image

Use helper script to spin the application

Pytorch install – Singularity containers

Example 04

```
SIF=/appl/local/containers/sif-images/lumi-pytorch-rocm-6.1.3-python-3.12-pytorch-v2.4.1.sif

rm -rf $wd/run-me.sh
cat > $wd/run-me.sh << EOF
#!/bin/bash -e

# Start conda environment inside the container
\$WITH_CONDA

# Run application
python -c 'import torch; print("I have this many devices:", torch.cuda.device_count())'

EOF
chmod +x $wd/run-me.sh

srun --jobid=$jobid -n1 --gpus 8 \
singularity exec \
 -B /var/spool/slurmd \
 -B /opt/cray \
 -B /usr/lib64/libcxi.so.1 \
-B $wd:/workdir \
 $SIF /workdir/run-me.sh
```

The container image to use: Pytorch 2.4.1 on top of ROCm 6.1.3

One could leverage a script to describe what is going to be executed inside the container.

This script has to load the container Conda environment. A special variable is set in the container to facilitate that.

Run as before.

Invoke singularity to start the container and execute the script created above.

AMD together we advance_

Controlling device visibility

- Controlling visibility
  - HIP_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'
  - ROCR_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())’
  - SLURM sets ROCR_VISIBLE_DEVICES
  - Implications of both ways of setting visibility – blit kernels and/or DMA

  - Considerations:
    - Does my app expects GPU visibility to be set in the environment?
    - Does my app expects arguments to define target GPUs
    - Does my app make any assumption on the device based on other information:
      - MPI rank
      - CPU-range
      - Auto-determined
    - How many processes using the same GPU:
      - Contention vs occupancy
      - Runtime scheduling limits
      - Increased scheduling complexity
      - Imbalance

Most Pytorch applications and driver scripts assume the GPU to be used corresponds to the local rank!!!

Testing affinity

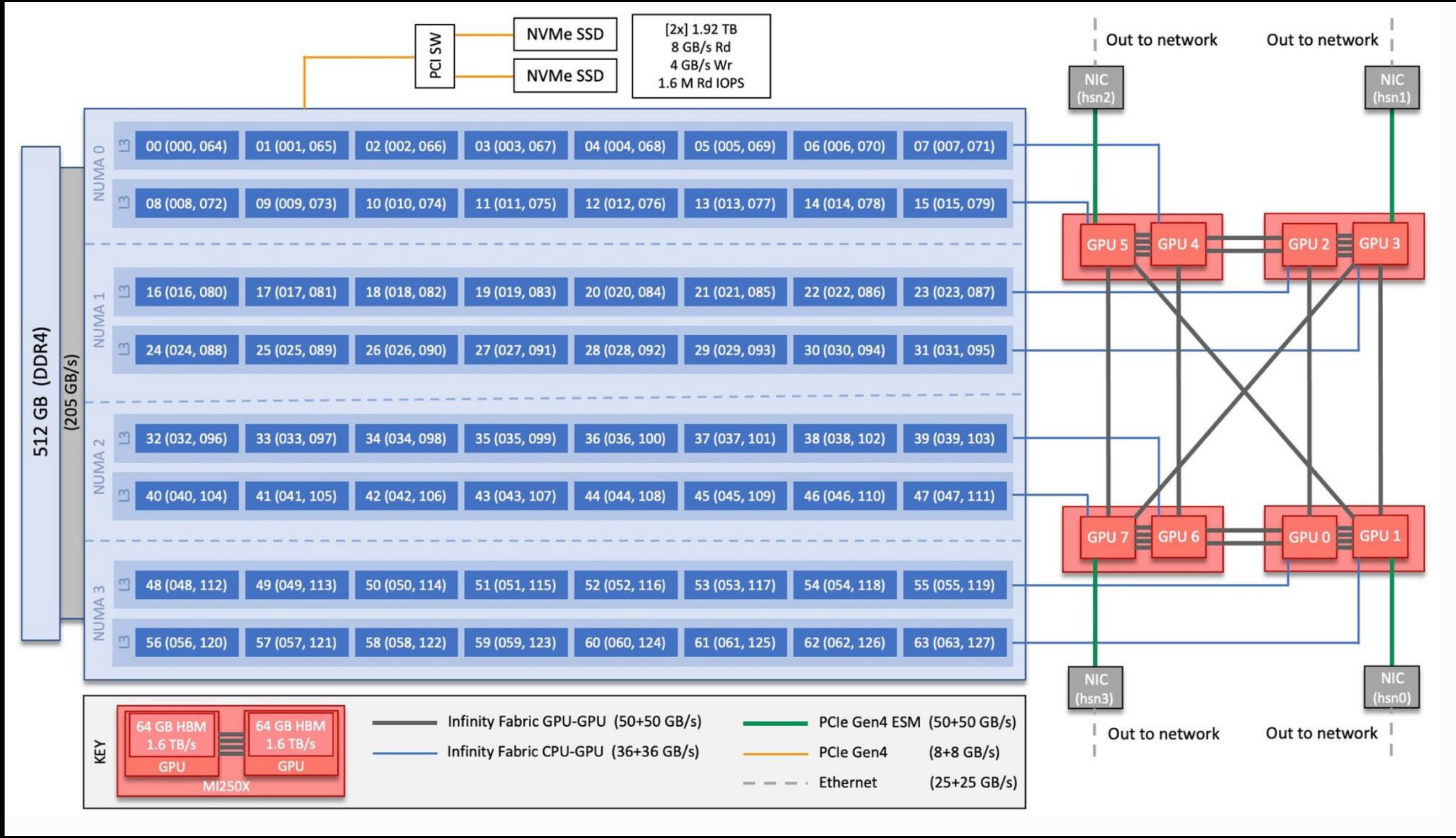
- What CPUs I have available and their NUMA domain?
  - lscpu
- What GPUs I have
  - rocm-smi –showtopo

NUMA node0 CPU(s):	0-15,64-79
NUMA node1 CPU(s):	16-31,80-95
NUMA node2 CPU(s):	32-47,96-111
NUMA node3 CPU(s):	48-63,112-127

GPU[0]	: (Topology) Numa Node: 3
GPU[0]	: (Topology) Numa Affinity: 3
GPU[1]	: (Topology) Numa Node: 3
GPU[1]	: (Topology) Numa Affinity: 3
GPU[2]	: (Topology) Numa Node: 1
GPU[2]	: (Topology) Numa Affinity: 1
GPU[3]	: (Topology) Numa Node: 1
GPU[3]	: (Topology) Numa Affinity: 1
GPU[4]	: (Topology) Numa Node: 0
GPU[4]	: (Topology) Numa Affinity: 0
GPU[5]	: (Topology) Numa Node: 0
GPU[5]	: (Topology) Numa Affinity: 0
GPU[6]	: (Topology) Numa Node: 2
GPU[6]	: (Topology) Numa Affinity: 2
GPU[7]	: (Topology) Numa Node: 2
GPU[7]	: (Topology) Numa Affinity: 2

Testing affinity

- ORNL topology - https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html



AMD
together we advance_

Testing affinity

- Check what SLURM is giving us:

```
srun -c 7 -N 2 -n 16 --gpus 16 \
 bash -c 'echo "$SLURM_PROCID -- GPUS $ROCR_VISIBLE_DEVICES -- $(taskset -p $$)”’ \
 | sort -n -k1
```

```
0 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54249's current affinity mask: fe
1 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54250's current affinity mask: fe00
2 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54251's current affinity mask: fe0000
3 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54252's current affinity mask: fe000000
4 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54253's current affinity mask: fe00000000
5 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54254's current affinity mask: fe0000000000
6 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54255's current affinity mask: fe000000000000
7 -- GPUS 0,1,2,3,4,5,6,7 -- pid 54256's current affinity mask: fe00000000000000
8 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110083's current affinity mask: fe
9 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110084's current affinity mask: fe00
10 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110085's current affinity mask: fe0000
11 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110086's current affinity mask: fe000000
12 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110087's current affinity mask: fe00000000
13 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110088's current affinity mask: fe0000000000
14 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110089's current affinity mask: fe000000000000
15 -- GPUS 0,1,2,3,4,5,6,7 -- pid 110090's current affinity mask: fe00000000000000
```

Example 05

Careful! Allocations do not follow GPU ranking!!

Testing affinity

- Check what SLURM is giving us:

```
srun -N 2 -n 16 --gpus 16 \
 --cpu-bind=mask_cpu:0xfe000000000000,0xfe00000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000 \
 bash -c 'echo "$SLURM_PROCID -- GPUS $ROCR_VISIBLE_DEVICES -- $(taskset -p $$)”’ \
 | sort -n -k1
```

```
0 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13819's current affinity mask: fe000000000000
1 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13820's current affinity mask: fe00000000000000
2 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13821's current affinity mask: fe0000
3 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13822's current affinity mask: fe000000
4 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13823's current affinity mask: fe
5 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13824's current affinity mask: fe00
6 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13825's current affinity mask: fe00000000
7 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13826's current affinity mask: fe0000000000
8 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94670's current affinity mask: fe000000000000
9 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94671's current affinity mask: fe00000000000000
10 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94672's current affinity mask: fe0000
11 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94673's current affinity mask: fe000000
12 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94674's current affinity mask: fe
13 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94675's current affinity mask: fe00
14 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94676's current affinity mask: fe00000000
15 -- GPUS 0,1,2,3,4,5,6,7 -- pid 94677's current affinity mask: fe0000000000
```

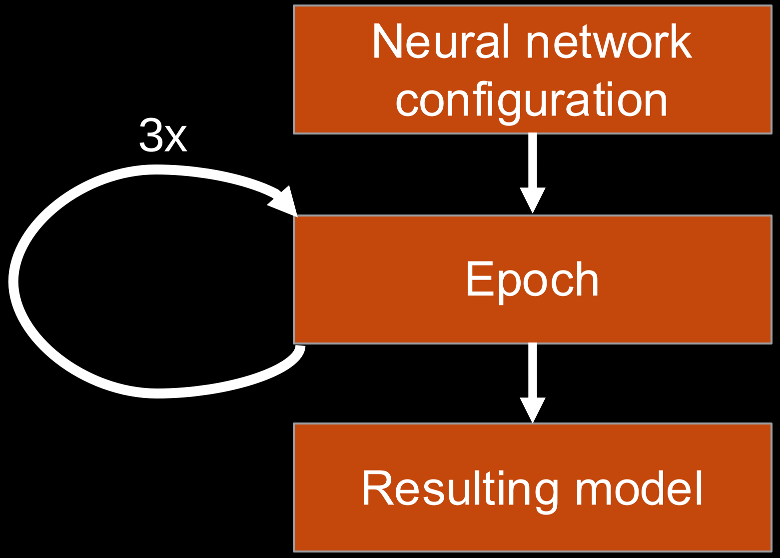
Great! CPUs are properly bound to the GPUs!

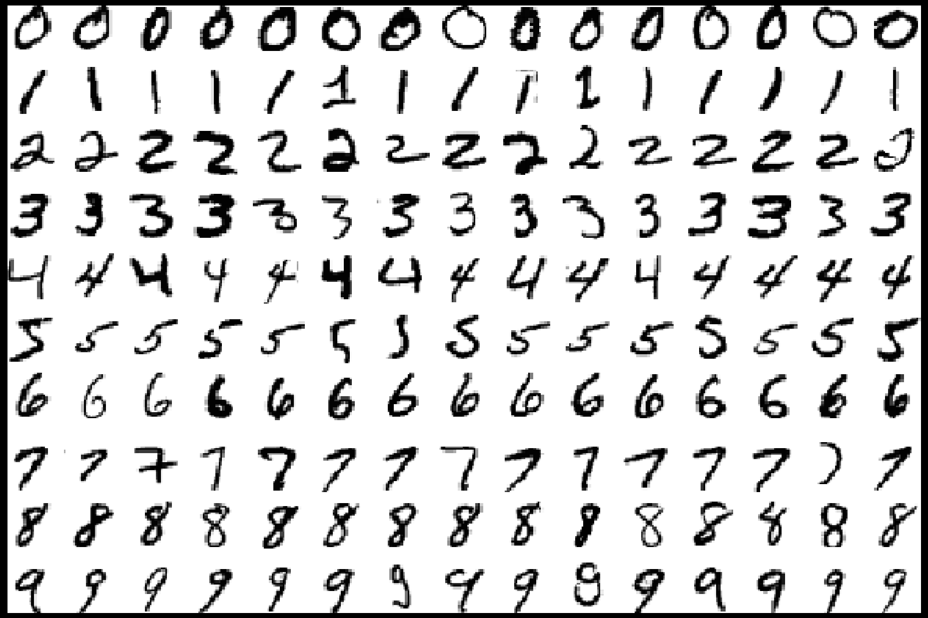
Example 05

AMD together we advance_

Pytorch example app – MNIST distributed learning

- Popular computer vision training dataset
- AI training iterate over epochs and a given sample batch is considered in each epoch
- Provided example runs over 3 epochs (properly trained models need much more than than!)
- MNIST training considers number images with different formats.





Pytorch example app – MNIST distributed learning

Examples 06

- What provides distributed capability:
  - Pytorch Distribute Data Parallel (DDP) – Batches of different data run concurrently
  - Other more sophisticated methods available
  - Frameworks like Deepspeed and Horovod can also enable distributed training.

```
import torch.distributed as dist
…
dist.init_process_group(
 backend='nccl',
 init_method='env://',
 world_size=int(os.environ['WORLD_SIZE']),
 rank=int(os.environ['RANK']))
```

Let’s use RCCL collectives library.

Well be learning about the distributed setting from the environment

Capture some env vars to adjust my distributed training

- … Epoch 0 Loss 0.148397 Global batch size 2048 on 16 ranks
- … Epoch 1 Loss 0.147906 Global batch size 2048 on 16 ranks
- … Epoch 2 Loss 0.147717 Global batch size 2048 on 16 ranks

Pytorch example app – MNIST distributed learning – RCCL

- RCCL should be set to use only high-speed-interfaces - Slingshot
- The problem one might see on startup:
-

```
NCCL error in: /workdir/pytorch-example/pytorch/torch/csrc/distributed/c10d/ProcessGroupNCCL.cpp:1269, unhandled system error, NCCL version 2.12.12
```

- Check error origin by setting RCCL specific debug environment variables:

Node has interfaces other than Slingshot

These are the correct ones.

```
export NCCL_DEBUG=INFO

NCCL INFO NET/Socket : Using [0]nmn0:10.120.116.65<0> [1]hsn0:10.253.6.67<0>
[2]hsn1:10.253.6.68<0> [3]hsn2:10.253.2.12<0> [4]hsn3:10.253.2.11<0>
NCCL INFO /long_pathname_so_that_rpms_can_package_the_debug_info/data/driver/rccl/src/init.cc:1292
```

- The fix:

```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
```

Point RCCL to use all 4 high-speed interfaces. It will know how to bind them based on the node topology.

Pytorch example app – MNIST distributed learning - script

Example 06

- What can/should I include in my start script:

```
if [ \$SLURM_LOCALID -eq 0 ] ; then
 rocm-smi
fi

export MIOPEN_USER_DB_PATH="/tmp/$(whoami)-miopen-cache-\$SLURM_NODEID"
export MIOPEN_CUSTOM_CACHE_DIR=\$MIOPEN_USER_DB_PATH

# Report affinity
echo "Rank \$SLURM_PROCID --> \$(taskset -p \$\$)"

# Start conda environment inside the container
\$WITH_CONDA

# Set interfaces to be used by RCCL.
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3

# Set environment for the app
export MASTER_ADDR=\$(python /workdir/get-master.py "\$SLURM_NODELIST")
export MASTER_PORT=29500
export WORLD_SIZE=\$SLURM_NPROCS
export RANK=\$SLURM_PROCID
export ROCR_VISIBLE_DEVICES=\$SLURM_LOCALID

# Run app
cd /workdir/mnist
python -u mnist_DDP.py --gpu --modelpath /workdir/mnist/model
```

Smoke test to confirm GPUs are available

Just-in-time compiles are a common technique in these applications. MIOpen leverages this functionality. Let's cache those builds in node-local storage instead of the default home folder. ROCm 6.2 may not need this.

Activate the container Conda environment that provides Pytorch

Point RCCL to use the high-speed network interfaces

Translate SLURM environment into something that Pytorch DDP understands

Run my model training

Pytorch example app – MNIST distributed learning - rocprof

Example 07

- Rocprof profiler client is the easiest way to get started with GPU profiling.
- It is available as part of the ROCm stack and, therefore, available in the containers
- It is seldomly useful to profile every single process/rank of your app:
  - Profilling more than needed = more potential profiling overhead
  - Misleading conclusions

Pick representative process/rank → Profile → Interpret and identify what is actionable

Command to prepend to my application instantiation

```
pcmd=''
if [ $RANK -eq 2 ] ; then
  pcmd='rocprof --hip-trace'
fi
```

We want to profile only for one rank – in this case rank #2

Run command as before except to the prepended profiling command

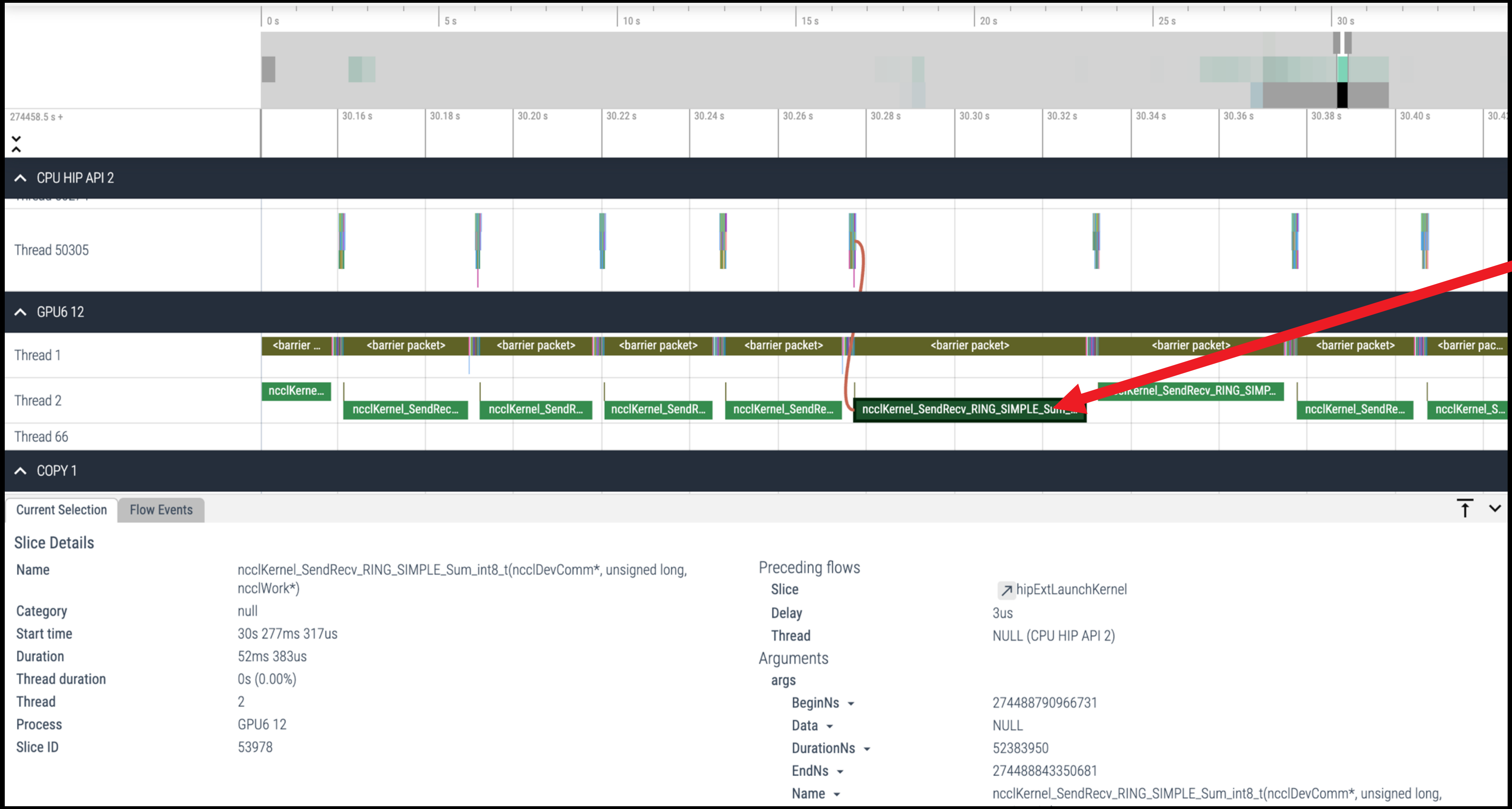
```
$pcmd python -u mnist_DDP.py --gpu --modelpath /workdir/mnist/model
```

More than one rank to be profiled? Use, –o myresults.$RANK.csv, to make sure sure there are no races generating the profile files

Pytorch example app – MNIST distributed learning - rocprof

- Bound by RCCL communication!

Example 07



Collectives kernels dominate profile

AMD
together we advance_

Comms are important! - RCCL AWS-CXI plugin

- LUMI, Frontier (and others) directly attaches AMD Instinct™ MI250x Accelerator to the Slingshot Network
  - Enable collectives computation on devices
  - Minimize the role of the CPU in the control path – expose more asynchronous computation opportunities
  - Lowest latency for network message passing is from GPU HBM memory

- CXI plugin is a runtime dependency. Requires: HPE Cray libfabric implementation
  - https://github.com/rocm/aws-ofi-rccl
  - 3-4x faster collectives
- **Included in the LUMI provided containers! If not using the LUMI containers make sure you have that in your environment:**

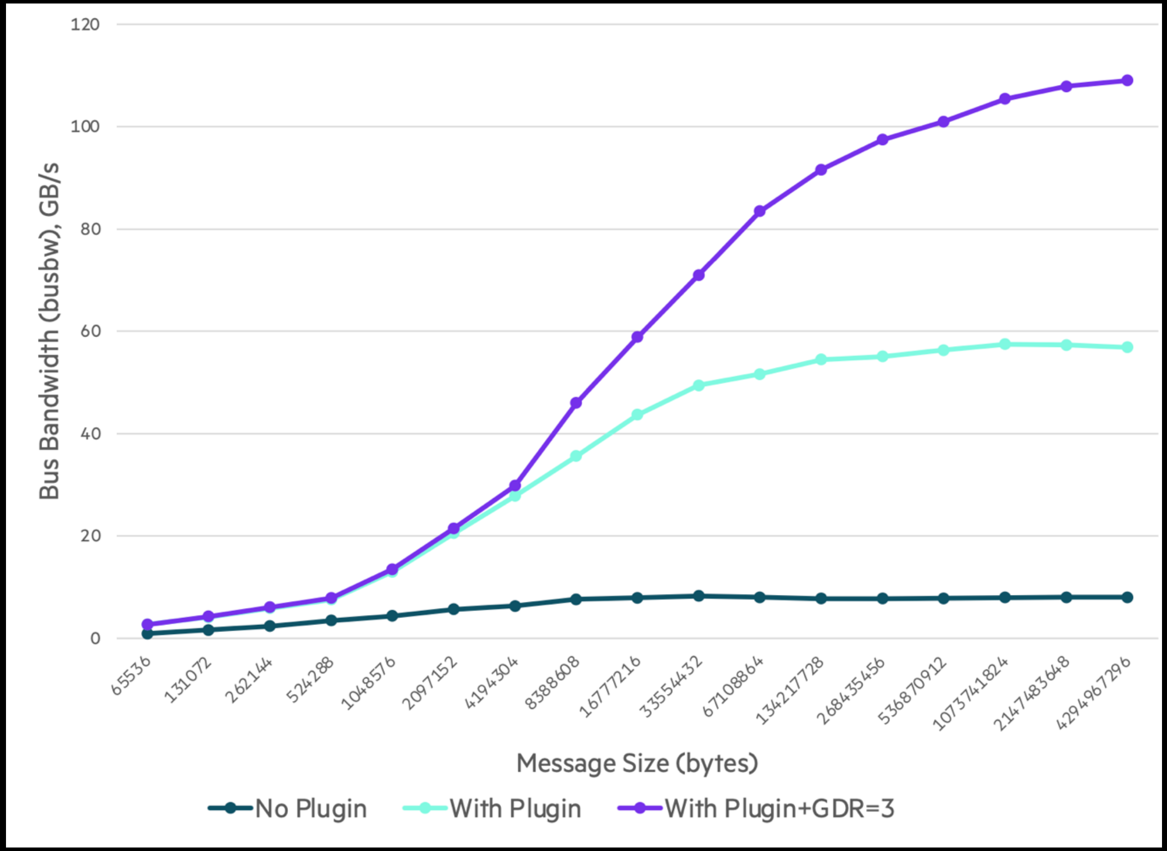
```
export NCCL_DEBUG=INFO
export NCCL_DEBUG_SUBSYS=INIT
# and search the logs for:
[0] NCCL INFO NET/OFI Using aws-ofi-rccl 1.4.0
```


AMD
together we advance_

Configuring RCCL environment (cont.)

- RCCL should be set configured to use GPU RDMA:
  - `export NCCL_NET_GDR_LEVEL=PHB`
- On upcoming ROCm versions (6.2) this won't be needed – it is default.
- Why should I spend time with all this?
  - 3-4x better bandwidth utilization with plugin
  - 2x better bandwidth utilization with RDMA
  - Can scale further!

- **Careful using external containers! You may need to be setting plugin yourself!**



AMD
together we advance_

Pytorch example app – MNIST distributed learning – Omnitrace

- Obtain more thorough trace information and visualization
  - https://github.com/AMDResearch/omnitrace
- Omnitrace install outside the container can be used
  - The host/container ROCm levels should match

```
module use module use /appl/local/containers/test-modules
module load rocm/6.1.3.lua omnitrace/1.12.0-rocm6.1.x
SIF=/appl/local/containers/sif-images/lumi-pytorch-rocm-6.1.3-python-3.12-pytorch-v2.4.1.sif
```

- Configuration file:
  - omnitrace-avail -G omnitrace.cfg –all
  - export OMNITRACE_CONFIG_FILE=/workdir/omnitrace-config.cfg
  - Override environment with command line arguments if needed

Example 08-09

Pytorch example app – MNIST distributed learning – Omnitrace

Example 08

- Sample – learn about the native stack trace along side GPU activity
- GPU activity is **never** sampled even in sampling mode
- Adjust configuration file according to the needs:

```
OMNITRACE_USE_SAMPLING = true
OMNITRACE_USE_ROCM_SMI = false
OMNITRACE_SAMPLING_CPUS = none
OMNITRACE_SAMPLING_GPUS = 2
```

Let's do sampling!
Not interested in sampling GPU hardware metrics (frequency, temperature…)
Not interested in sampling CPU hardware metrics
Targeting GPU #2 only used by Rank #2

- Execution similar to rocprof:

```
pcmd=''
if [ $RANK -eq 2 ] ; then
  pcmd='omnitrace-sample -- '
fi
$pcmd python -u mnist_DDP.py --gpu --modelpath /workdir/mnist/model
```

Prepend omnitrace sampling driver for rank #2

- We need to add a few more bindings to singularity:

```
srun --jobid=$jobid -N $((Nodes)) -n $((Nodes*8)) --gpus $((Nodes*8)) --cpu-bind=mask_cpu:$MYMASKS \
  singularity exec \
    …\
    -B $wd:/workdir \
    -B $OMNITRACE_dir/omnitools \
    -B /usr/lib64/libpciaccess.so.0 \
    $SIF /workdir/run-me.sh
```

Make omnitrace available in the container
Omnitrace does PCIe info loading – so we need to enable that

- …and make sure the environment inside the container is set accordingly:

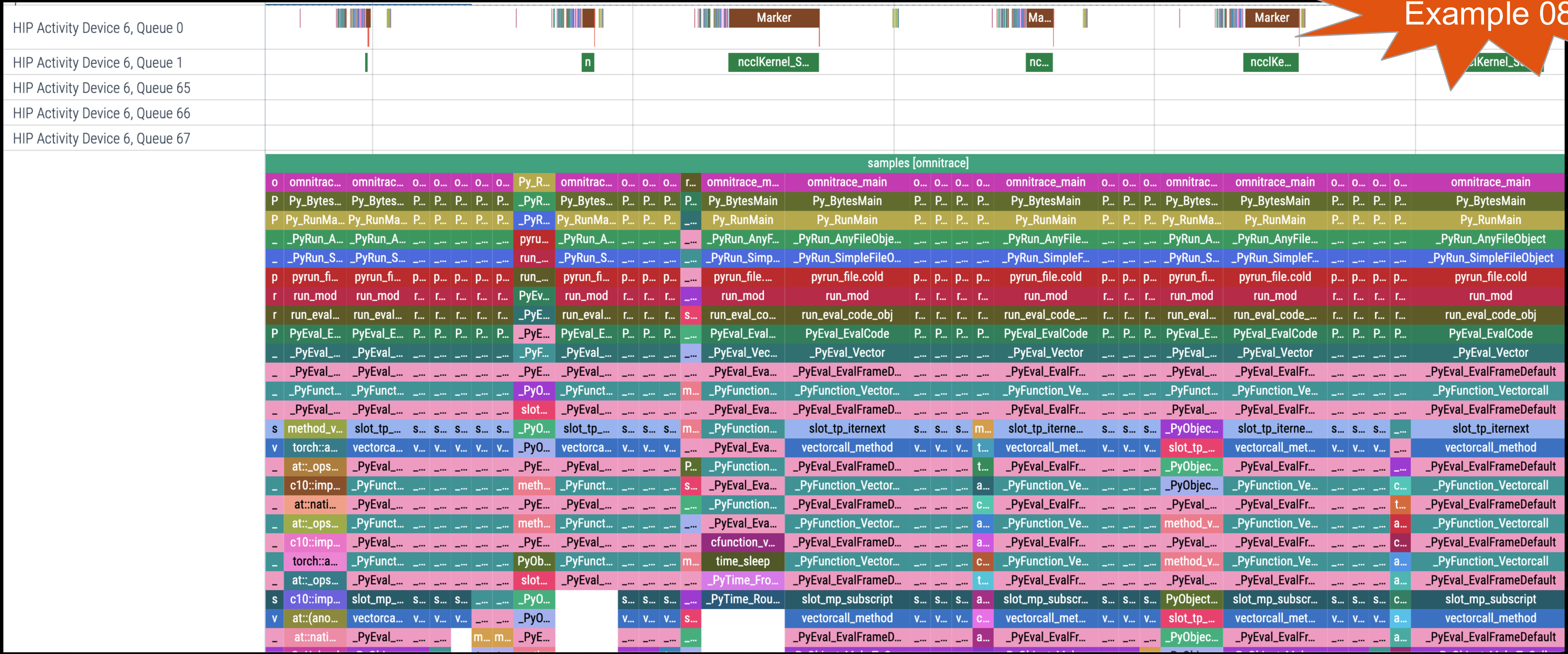
```
export PATH=$OMNITRACE_dir/bin:$PATH
export LD_LIBRARY_PATH=$OMNITRACE_dir/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$OMNITRACE_dir/lib/python/site-packages:$LD_LIBRARY_PATH
```

Makes sure all Omnitrace bits are available in my environment.

Pytorch example app – MNIST distributed learning – Omnitrace

Example 08

- Native stack flame graph:



Pytorch example app – MNIST distributed learning – Omnitrace

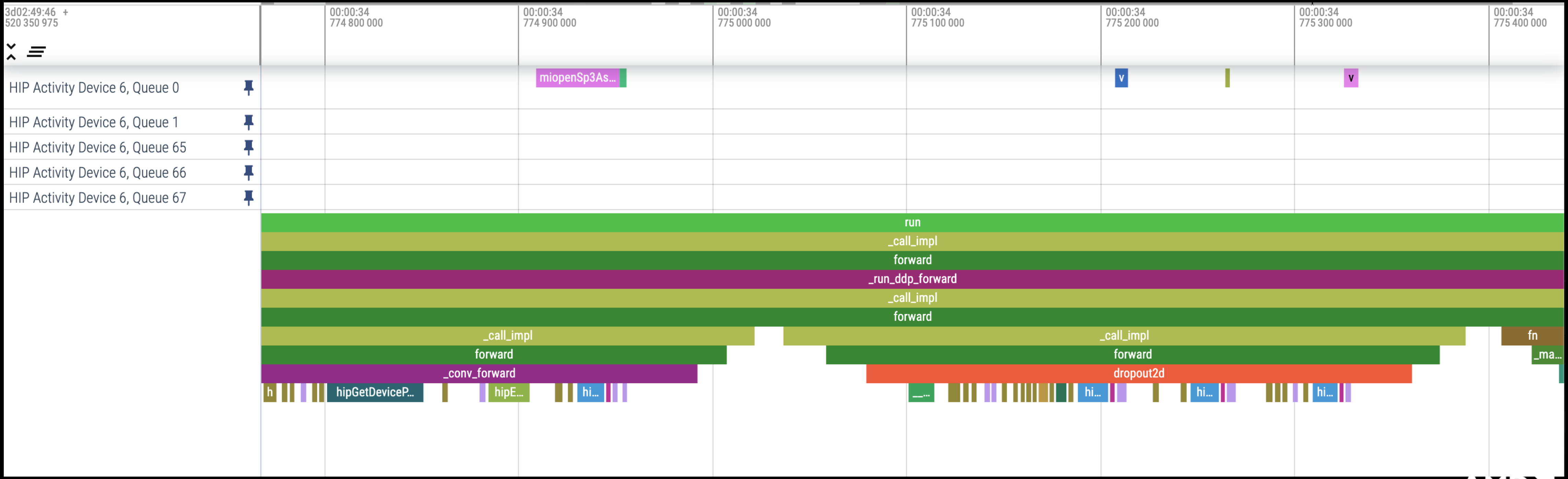
- Sampling the Python and C/C++ parts of the code – omnitrace-python

Example 9

```
if [ \$RANK -eq 2 ] ; then
 omnitrace-python-3.10 -- mnist_DDP.py --gpu --modelpath /workdir/mnist/model
else
 python -u mnist_DDP.py --gpu --modelpath /workdir/mnist/model
fi
```

Omnitrace expects the Python script as opposed to the Python executable

Match relevant python version



AMD together we advance_

Pytorch example app – MNIST distributed learning – Omniperf

Example 10

- Obtain detailed kernel performance counters
  - https://github.com/AMDResearch/omniperf

```
module use module use /appl/local/containers/test-modules
module load rocm/6.1.3.lua omniperf/2.1.0
```

- Virtual environment is used to extend the existing Python environment inside the container.
- Omniperf needs replaying the application many times
  - Could be challenging to profile individual ranks as all need replaying.

```
if [ $RANK -eq 0 ] ; then
 omniperf profile -n myprof --device 0 --roof-only -- $(which python) -u mnist_DDP.py ...
else
 for i in {1..4} ; do
  sleep 10
  python -u mnist_DDP.py --gpu --modelpath /workdir/mnist/model
 done
fi
```

Collect roofline profile for rank #0 that uses device #0

Replay the app on the other ranks as many times as needed (5 times for roofline)

AMD together we advance_

Pytorch example app – MNIST distributed learning – Omniperf

Example 10

- Analyze in or outside the container):
  - omniperf analyze -p workloads/pytorch/MI200/ --gui

Select kernel of interest

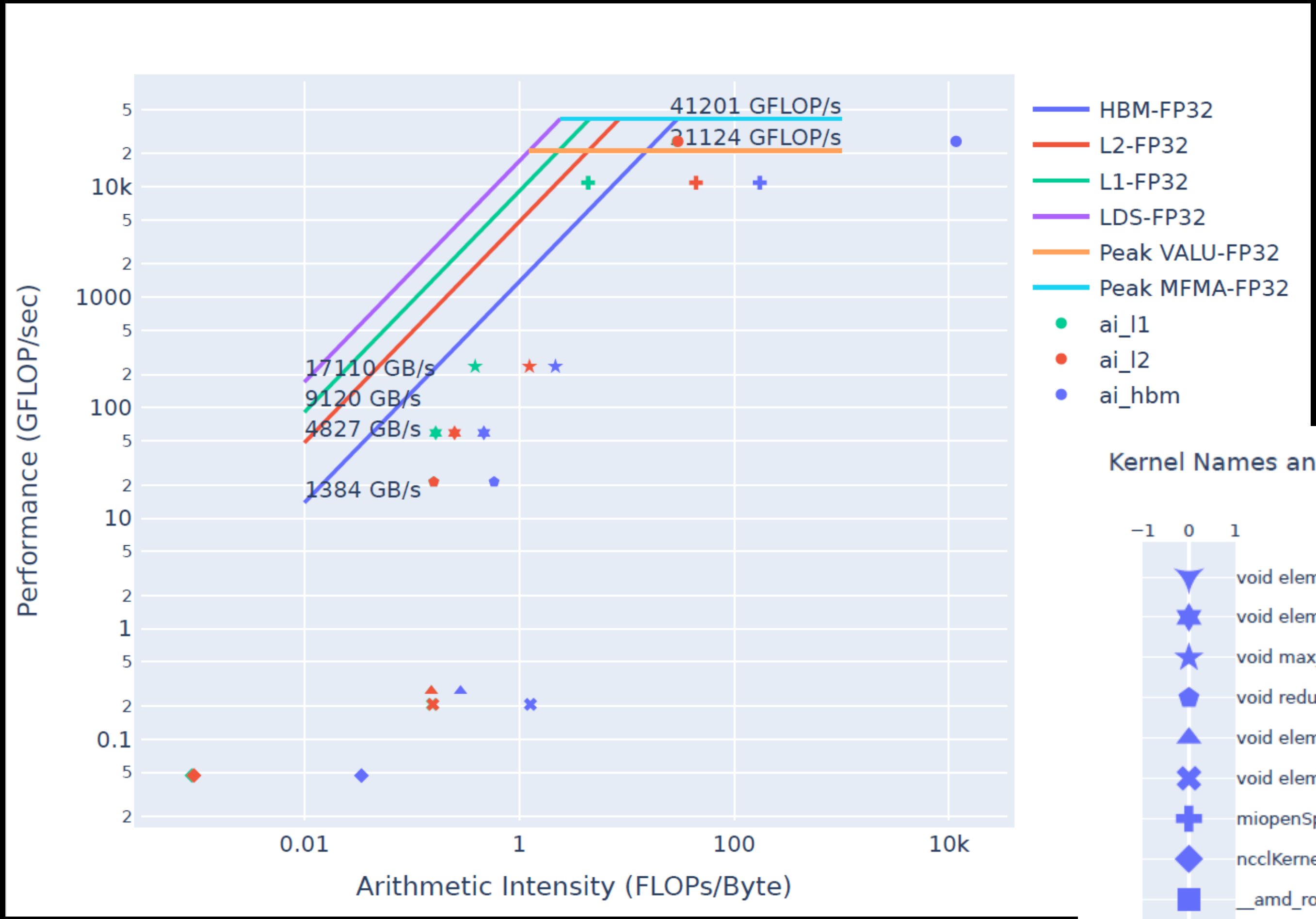
AMD together we advance_

Pytorch example app – MNIST distributed learning – Omniperf

- Analyze in or outside the container):
  - Roofline PDFs

Example 10

Kernel names



Leveraging framework profiler infrastructure

Example 11

- AI frameworks typically provide hooks for developers to gather profiling information
- An example with Pytorch:

```
from torch.profiler import profile, record_function, ProfilerActivity

for epoch in range(args.epochs):

    prof = None
    if epoch == 3:
        print("Starting profile...")
        prof = profile(activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA])
        prof.start()

    for imgs, labels in dataloader:
        with torch.amp.autocast('cuda',enabled=args.amp):
            imgs, labels = imgs.cuda(), labels.cuda()
            outputs = model(imgs)
        loss = criterion(outputs, labels)
        loss = scaler.scale(loss)
        loss.backward()
        scaler.step(optimizer)
        scaler.update()

    if prof:
        prof.stop()
        prof.export_chrome_trace("trace.json")
```

Invoke the profiler

Enable profiling for epoch number 3

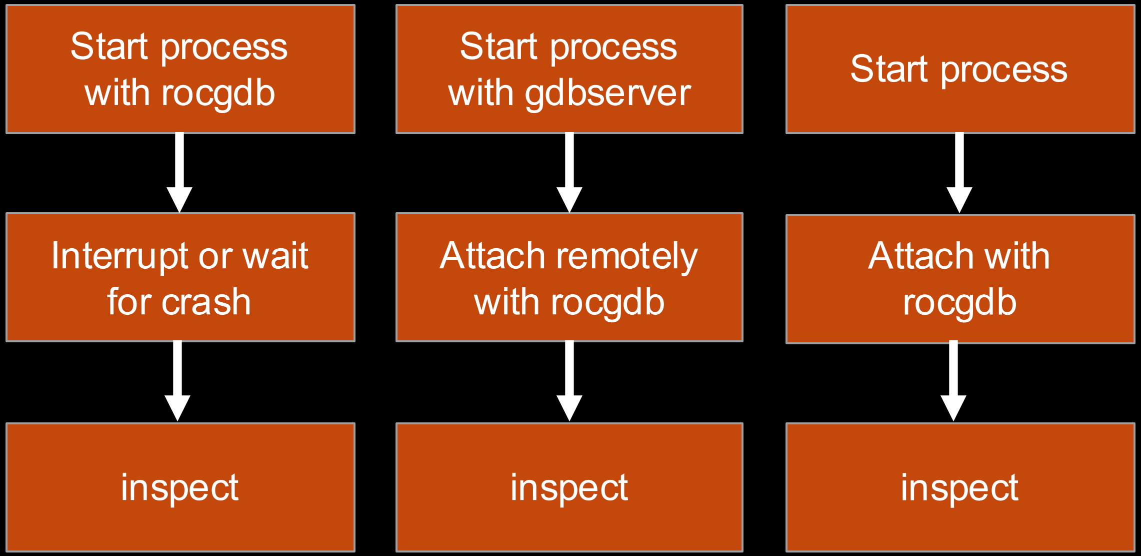
Training for an epoch

Finish profiling and generate trace

Trace file can be viewed in Perfetto UI tool

Pytorch example app – MNIST distributed learning – Rocgdb

- Debugging requires proper driver support
  - Can't run debugger effectively from incompatible containers
  - Use system ROCm for roccgdb
- Two main use cases to use
  - Connecting into a hanging process
  - Progress up to breakpoint or segfault
- ROCm provides rocgdb – you may need your own gdbserver.
- Using gdbserver is possible
  - gdbserver can be issued conveniently as a profile tool
  - Launch with:
    - `gdbserver --once $(hostname):12345 ./my_command`
  - Attach with
    - `rocgdb -x gdb.commands ./my_command`
  - Leverage gdb commands file to automate startup
    - `target remote target_host:12345`



3 different workflows to choose from!

If interested in stepping into GPU code you should avoid gdbserver

Disclaimer

Questions?

AMD