



Tools in action

An example with Pytorch

George Markomanolis, Samuel Antao

Comprehensive General LUMI Course
Espoo, April 23–26th 2024

AMD 
together we advance_

slides on LUMI in `/project/project_465001098/Slides/AMD/`

hands-on exercises: https://hackmd.io/@gmarkoma/lumi_finland

hands-on source code: `/project/project_465001098/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!

Examples in: https://hackmd.io/@gmarkoma/lumi_finland#MNIST-example

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, many others
 - APEX – multiprecision library
 - <https://github.com/ROCmSoftwarePlatform/apex>



Pytorch install – our base environment

```
module purge
```


```
module load CrayEnv
```

```
module load PrgEnv-cray/8.4.0
```


```
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 /pfs/lustrep2/projappl/project_462000125/samantao-public/mymodules
```

```
module load rocm/5.6.1.lua
```

We will be using 5.6.1 and patched 5.5.1 in these examples.
ROCm 5.6.1 brings fixes but ROCm 5.5.1 is more compatible with the current LUMI driver



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

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

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.

```
pip3 install -t $PWD/pip-installs --pre torch==2.2.2+rocm5.6 --index-url https://download.pytorch.org/whl/
```

```
Collecting torch==2.2.2+rocm5.6
  Downloading https://download.pytorch.org/whl/rocm5.6/torch-2.2.2%2Brocm5.6-cp310-cp310-linux_x86_64.whl (1570.8 MB)
----- 1.6/1.6 GB 88.0 MB/s eta 0:00:01
```

```
PYTHONPATH=$PWD/pip-installs \
```

```
srunc --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
```

Make the frershly install Pytorch available to your Python runs

Should yield the number of GCDs in the node.



Pytorch install – virtual environments

- 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==2.2.2+rocm5.6 --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())'
```

If error with pip version:

```
pip3 install --upgrade pip
```


Pytorch install – conda environment

- 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-py310_24.1.2-0-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=3.10
```

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

Create and activate a conda environment to install Pytorch based on Python 3.10

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

```
pip3 install --pre torch==2.2.2+rocm5.6 --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 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/8.4.0 gcc/10.3.0
```

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

```
git clone -b v2.2.2 --recursive \
```

```
https://github.com/pytorch/pytorch_pytorch-source
```

```
# Create and activate conda environment to manage the install
```

```
conda create -y -n pytorch-from-source python=3.10
```

```
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 installation 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 \
```

```
RCCL_PATH=$ROCM_PATH/rccl \
```

```
RCCL_DIR=$ROCM_PATH/rccl/lib/cmake \
```

```
hip_DIR=${ROCM_PATH}/hip/cmake/ \
```

```
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 /usr/lib64/libjansson.so.4 \
  -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



```
SIF=/apl/local/containers/sif-images/lumi-pytorch-rocm-5.6.1-python-3.10-pytorch-v2.2.2.sif
```

The container image to use:
Pytorch 2.2.2 on top of ROCm 5.6.1

```
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
```

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

```
srunc --jobid=$jobid -n1 --gpus 8 \
```

```
singularity exec \
```

```
-B /var/spool/slurmd \
```

```
-B /opt/cray \
```

```
-B /usr/lib64/libcxi.so.1 \
```

```
-B /usr/lib64/libjansson.so.4 \
```

```
-B $wd:/workdir \
```

```
$SIF /workdir/run-me.sh
```

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.

Controlling device visibility

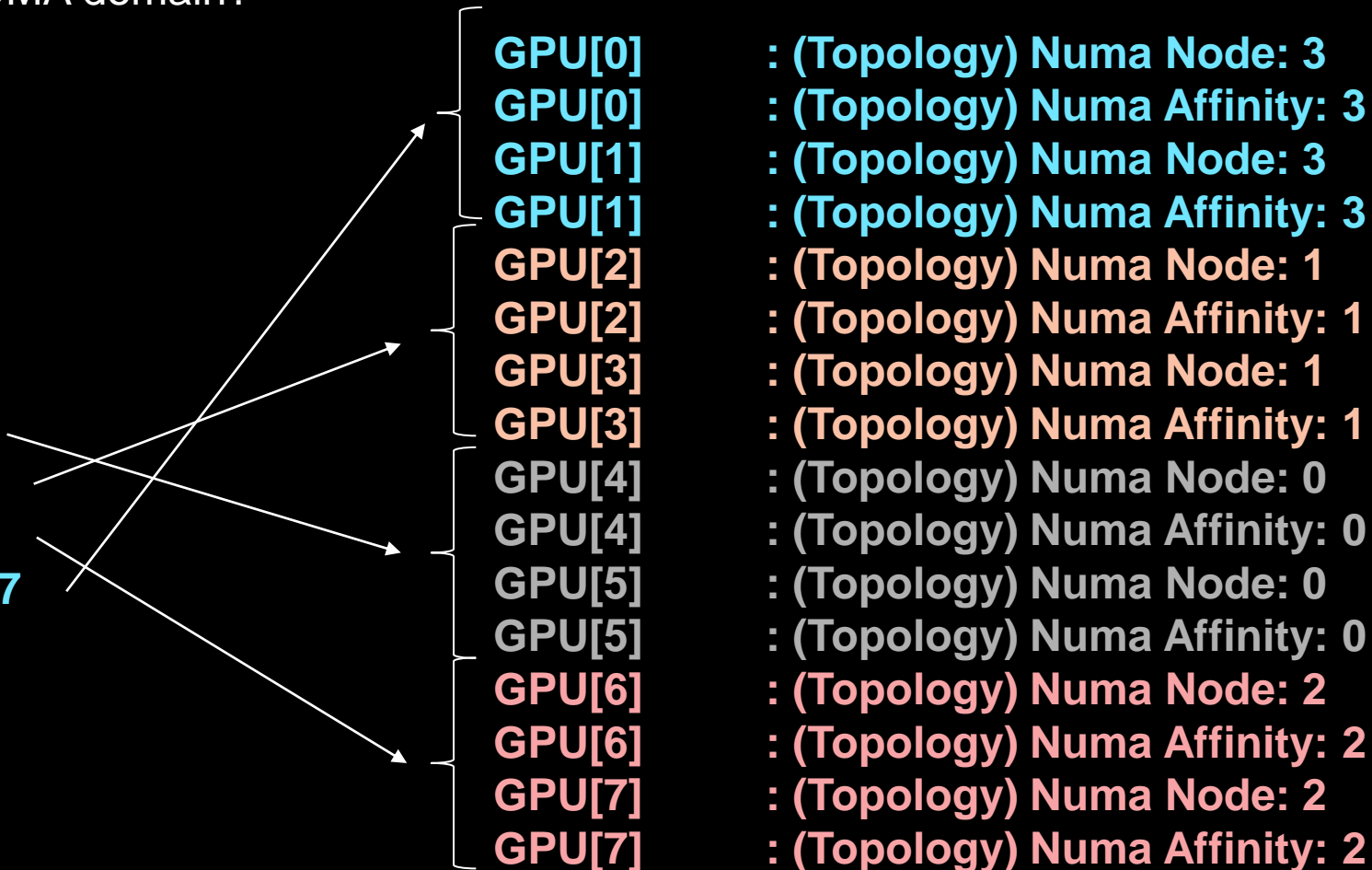
- Controlling visibility
 - `HIP_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'`
 - `ROCR_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'`
 - SLURM sets `ROCR_VISIBLE_DEVICES`
 - Implications of both ways of setting visibility – blit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

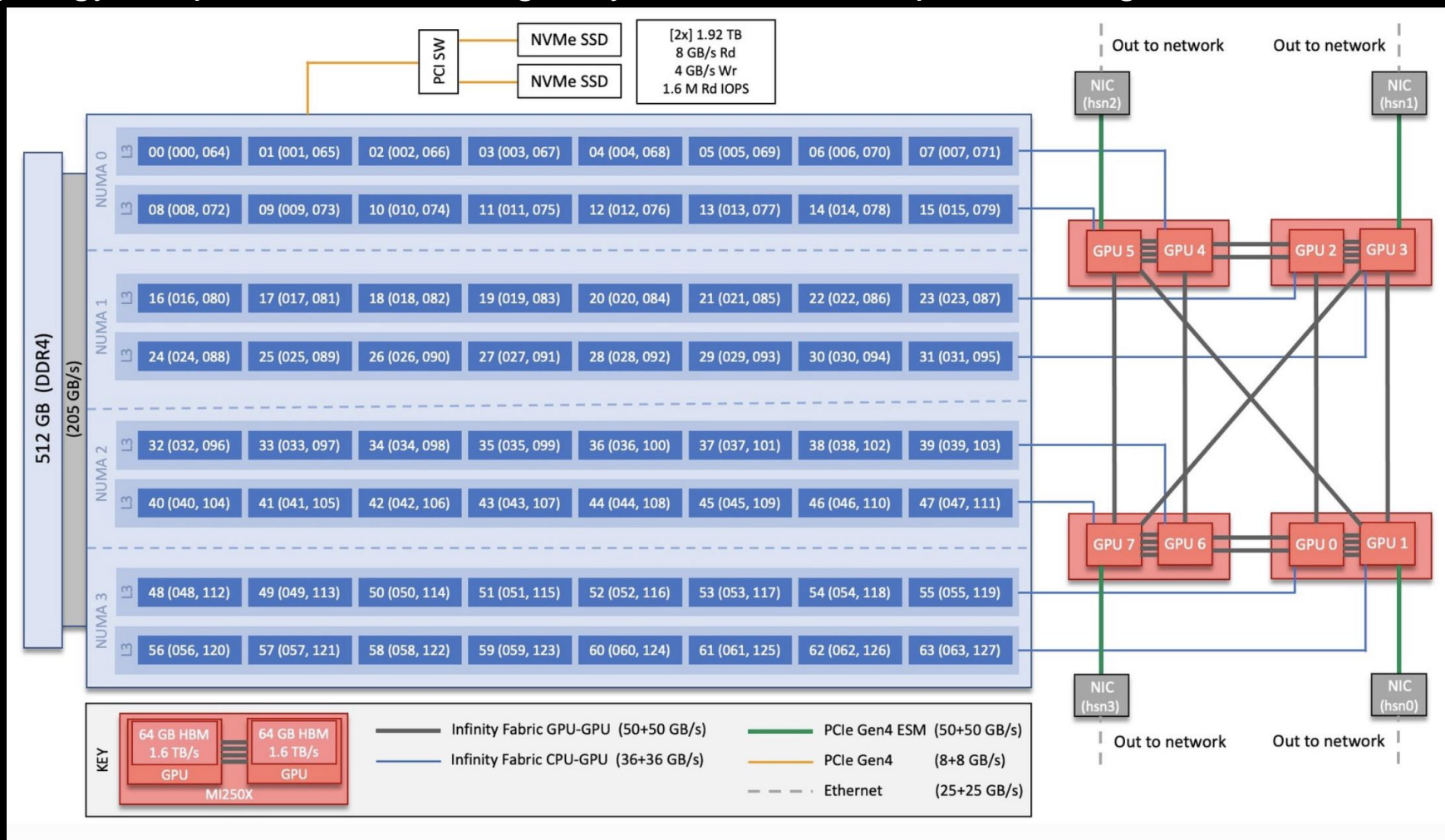
- What CPUs I have available and their NUMA domain?
 - lscpu
- What GPUs I have
 - rocm-smi -showtopo

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



Testing affinity

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



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 06

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:0xfe000000000000,0xfe000000000000,0xfe0000,0xfe000000,0xfe,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
```

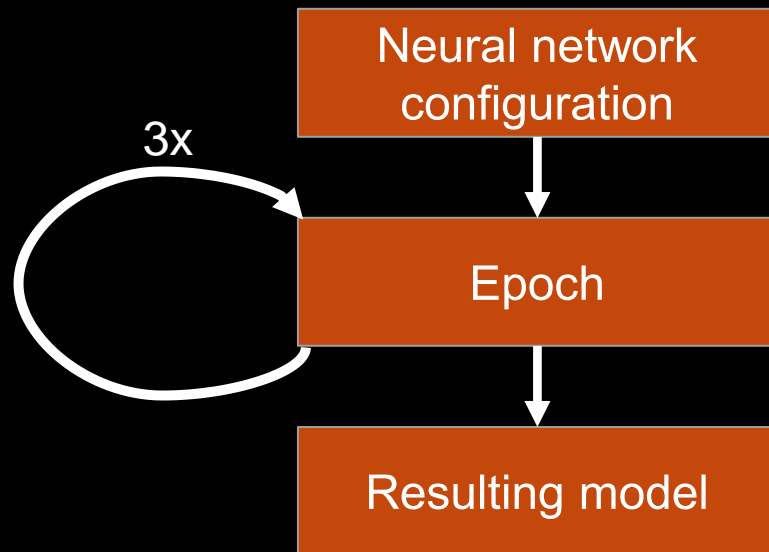


Great! CPUs are properly bound to the GPUs!

Example 06

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

- 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.

We'll 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:

```
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
```

Node has interfaces other than Slingshot

These are the correct ones.

- 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



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

Smoke test to confirm GPUs are available

```
if [ \${SLURM_LOCALID} -eq 0 ]; then
```

```
rocm-smi
```

```
fi
```

```
export MIOOPEN_USER_DB_PATH="/tmp/${whoami}-miopen-cache-${SLURM_NODEID}"
```

```
export MIOOPEN_CUSTOM_CACHE_DIR=${MIOOPEN_USER_DB_PATH}
```

```
# Report affinity
```

```
echo "Rank \${SLURM_PROCID} --> \${taskset -p \${SLURM_PROCID}}"
```

```
# 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
```

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.

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

- 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:
 - Profiling more than needed = more potential profiling overhead
 - Misleading conclusions



Example 08



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

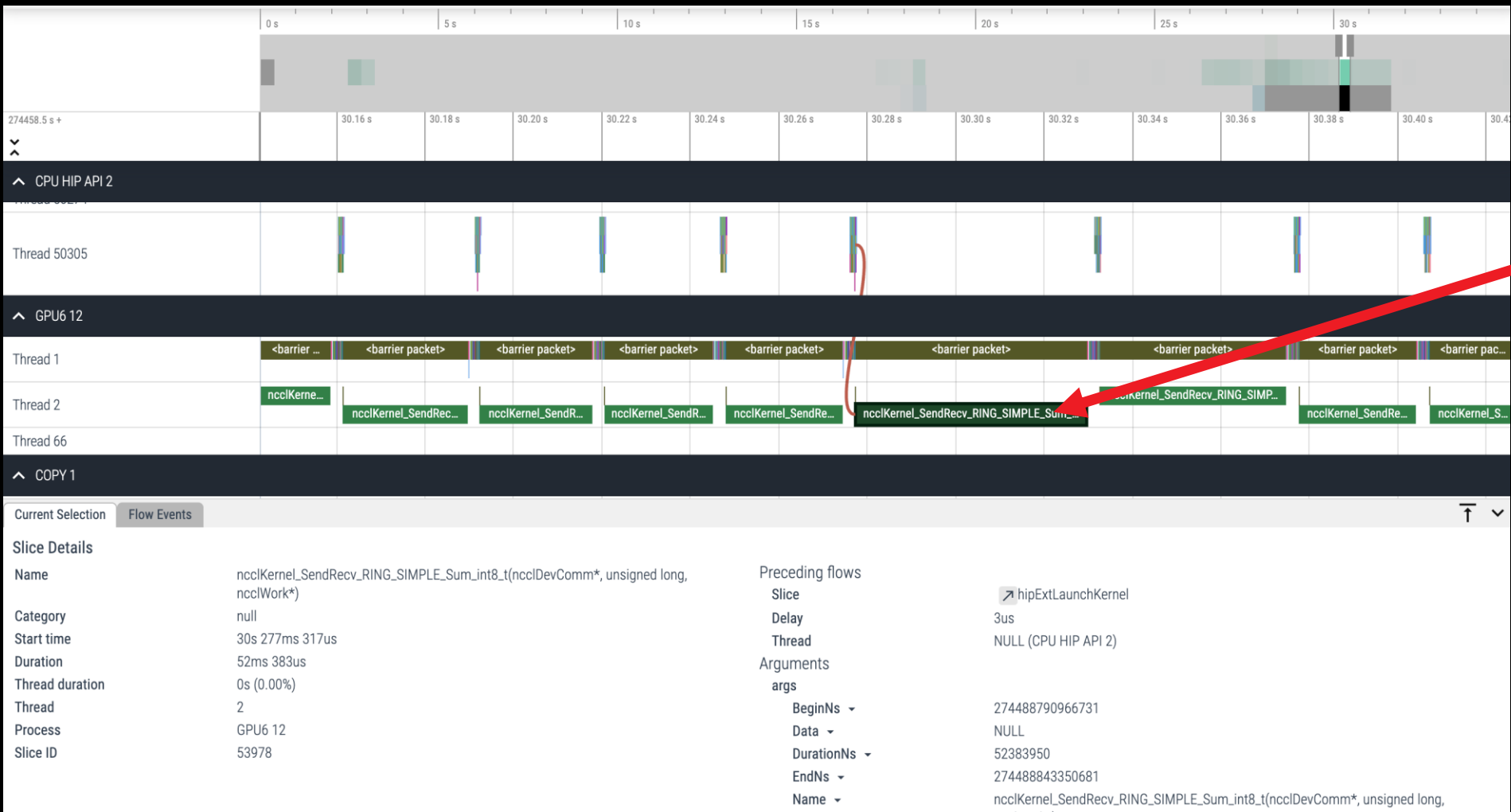
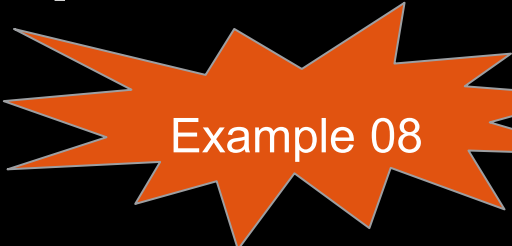
Run command as before except to the prepended profiling command

```
$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 there are no races generating the profile files

Pytorch example app – MNIST distributed learning - rocprof

- Bound by RCCL communication!



Collectives kernels dominate profile

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/ROCmSoftwarePlatform/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
```


Pytorch example app – MNIST distributed learning – Omnitrace

- Obtain more thorough trace information and visualization
 - <https://github.com/AMDRResearch/omnitrace>
 - OpenSUSE 15.4 build mainly compatible with LUMI environment
- Omnitrace is sensitive to ROCm driver level limitations
 - We need to rollback to ROCm 5.5.x for LUMI (ROCm 5.2.3 driver)
 - The container has to match this ROCm level

```
module use /pfs/lustrep2/projappl/project_462000125/samantao-public/mymodules
```

```
module load rocm/5.5.3 omnitrace/1.11.2-rocm-5.5.x
```

```
SIF=/appl/local/containers/sif-images/lumi-pytorch-rocm-5.5.1-python-3.10-pytorch-v2.0.1.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 09-10

Pytorch example app – MNIST distributed learning – Omnitrace



Example 09

- 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

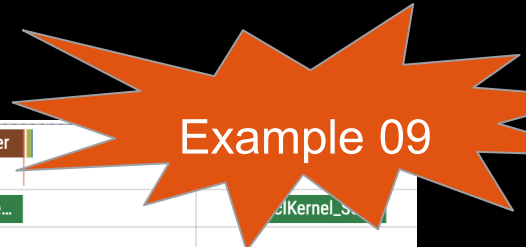
- ...and make sure the environment inside the container is set accordingly:

```
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.

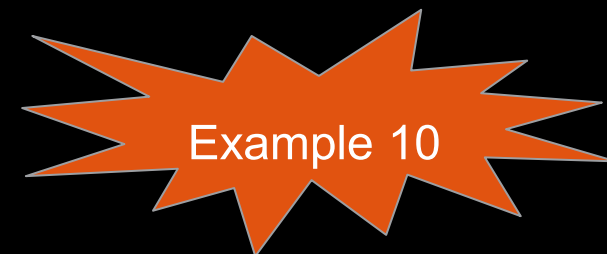
Pytorch example app – MNIST distributed learning – Omnitrace

- Native stack frame graph:

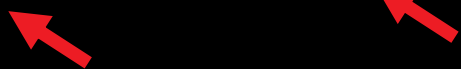


Pytorch example app – MNIST distributed learning – Omnitrace

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



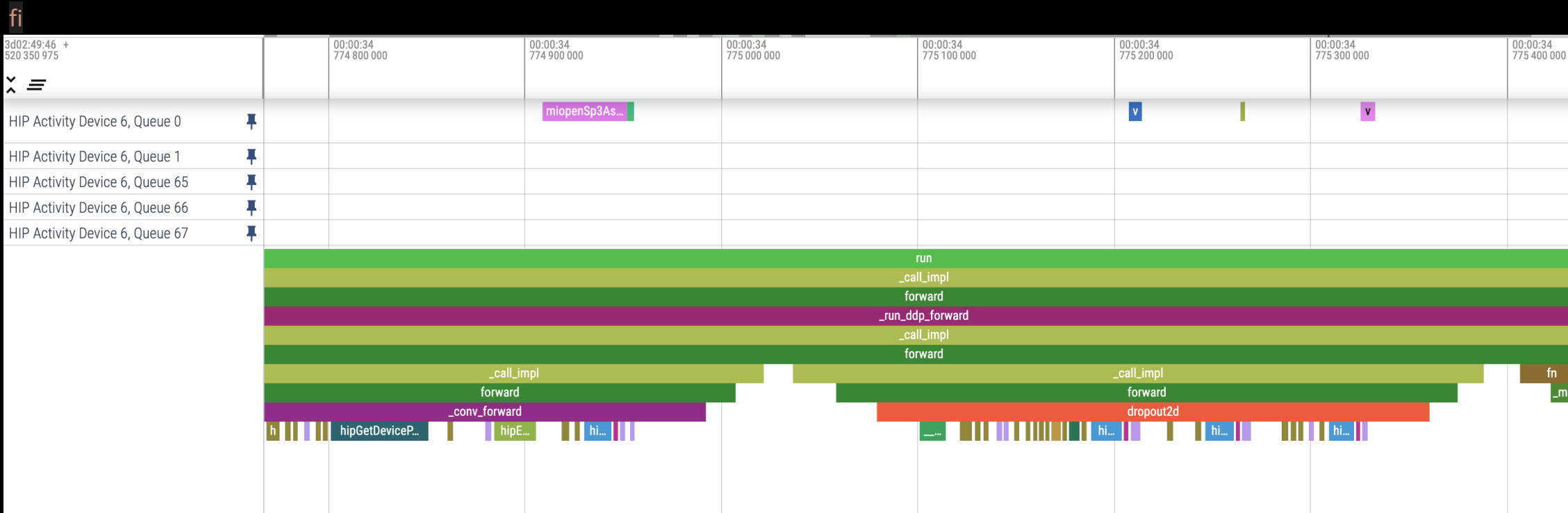
```
if [ \${RANK} -eq 2 ] ; then
  omnitrace-python-3.10 -- mnist_DDP.py --gpu --modelpath /workdir/mnist/model
else
```



Omnitrace expects the Python script as opposed to the Python executable

Match relevant python version

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



Pytorch example app – MNIST distributed learning – Omniperf



Example 11

- Obtain detailed kernel performance counters
 - <https://github.com/AMDRResearch/omniperf>

```
module use /pfs/lustrep2/projappl/project_462000125/samantao-public/mymodules
module load rocm/5.5.3 omniperf/1.0.10-rocm-5.5.x
```

- 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)

Pytorch example app – MNIST distributed learning – Omnipperf



- Analyze in or outside the container):
 - `omnipperf analyze -p workloads/pytorch/mi200/ --gui`

Menu ▾ NORMALIZATION: GCD: DISPATCH FILTER: TOP N:

KERNELS:

Empirical Roofline Analysis (FP32/FP64)

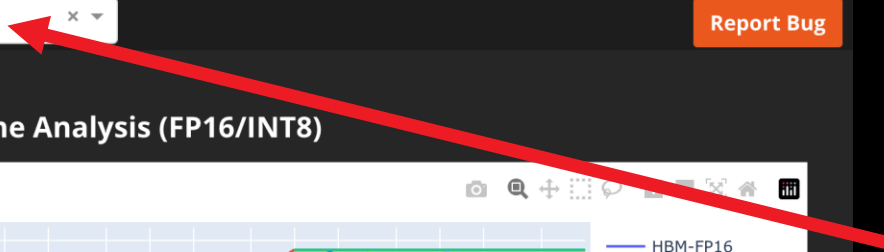
Legend:

- HBM-FP32
- L2-FP32
- L1-FP32
- LDS-FP32
- Peak VALU-FP32
- Peak MFMA-FP32
- ai_l1
- ai_l2
- ai_hbm

Empirical Roofline Analysis (FP16/INT8)

Legend:

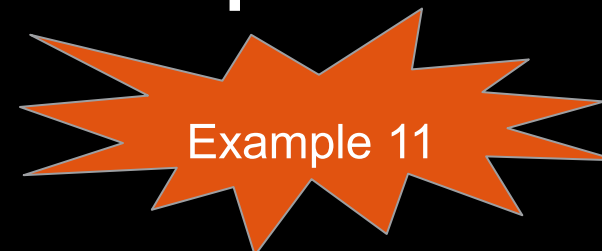
- HBM-FP16
- L2-FP16
- L1-FP16
- LDS-FP16
- Peak MFMA-FP16
- ai_l1
- ai_l2
- ai_hbm
- HBM-I8
- L2-I8
- L1-I8
- LDS-I8
- Peak MFMA-I8



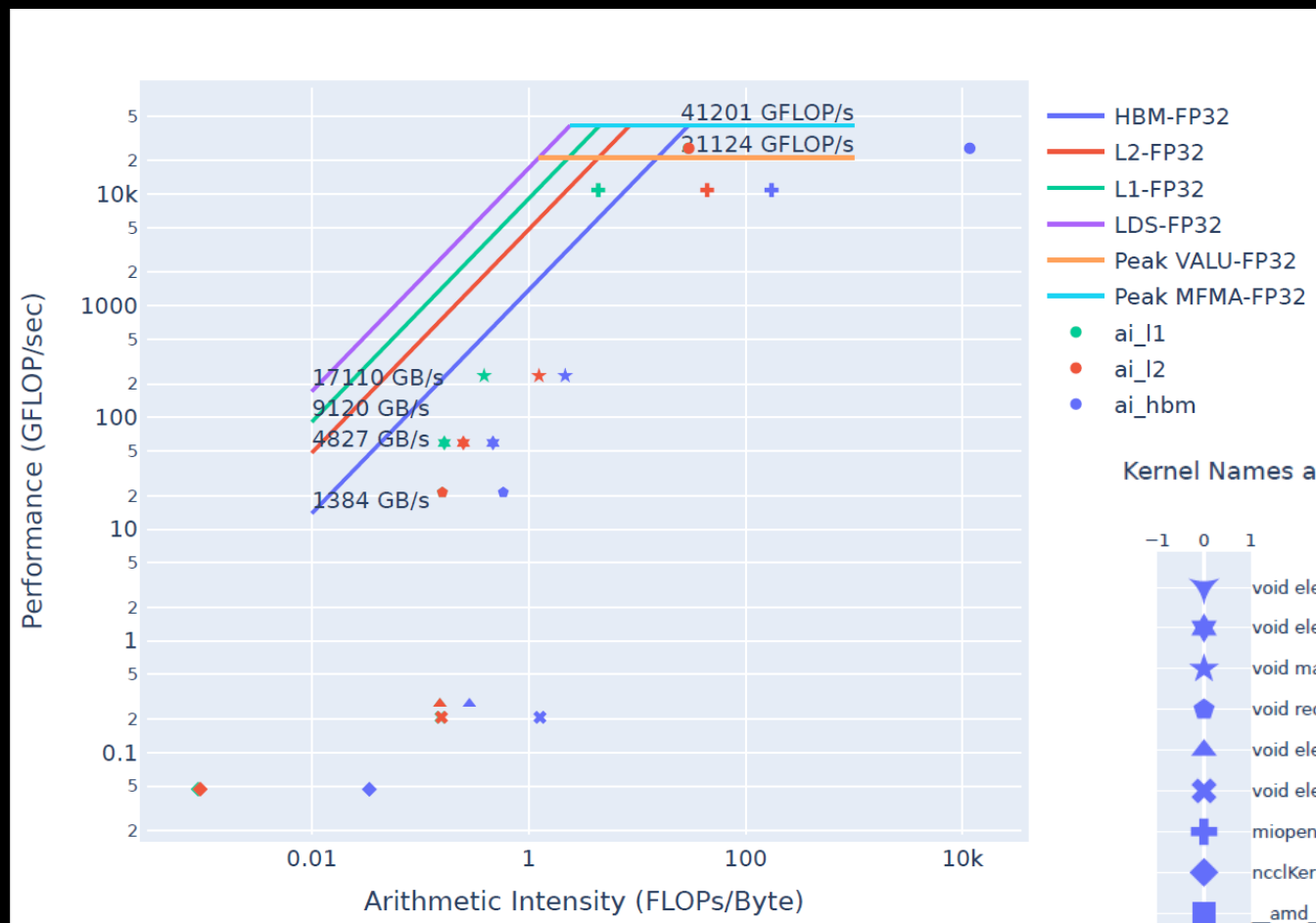
Select kernel of interest

Pytorch example app – MNIST distributed learning – Omnipperf

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



Kernel names

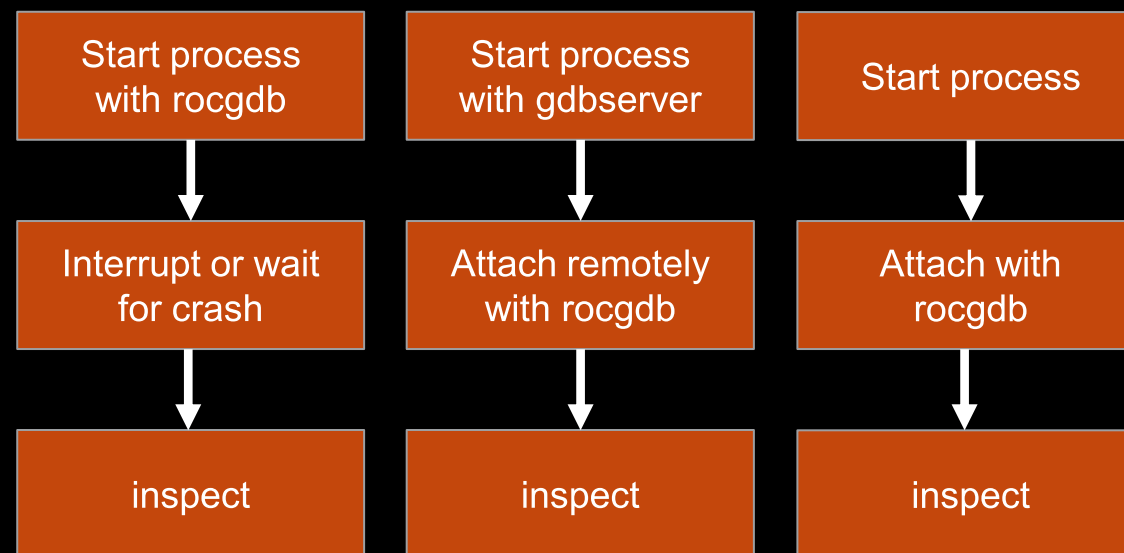


Kernel Names and Markers

Marker	Kernel Name
▼	void elementwise_kernel<launch_clamp_scalar(Scalar, Scalar, lambda(operator)) lambda(operator)) lambda(float)Array >(int, launch_clamp...
★	void elementwise_kernel<128, 4, gpu_kernel_impl(CUDAFunctor_add lambda(int)(int, gpu_kernel_impl(CUDAFunctor_add lambda(int))...
☆	void max_pool_backward_nchw<float, float>(float const*, long const*, int, long, long, long, int, int, int, int, int, int, int, float*)
⬢	void reduce_kernel<512, 1, ReduceOp
▲	void elementwise_kernel<BUnaryFunctor, Array >(int, BUnaryFunctor<float, float, float, MulFunctor >, Array<char*, 2>)
✕	void elementwise_kernel<AUnaryFunctor, Array >(int, AUnaryFunctor<float, float, float, MulFunctor >, Array<char*, 2>)
+	miopenSp3AsmConv_v21_1_3_gfx9_fp32_stride1.kd
◆	ncclKernel_SendRecv_RING_SIMPLE_Sum_int8_t(ncclDevComm*, unsigned long, ncclWork*)
■	__amd_rocclr_fillBufferAligned.kd
●	miopenSp3AsmConv_v21_1_3_gfx9_fp32_stride1_group.kd

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
 - Branches for given ROCm releases: e.g. rocm-5.2.x
 - <https://github.com/ROCm/ROCgdb>
- Two main use cases to use
 - Connecting into a hanging process
 - Progress up to breakpoint or segfault
- ROCm provides rocdbg – 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
 - `rocdbg -x gdb.commands ./my_command`
 - Leverage gdb commands file to automate startup
 - `target remote target_host:12345`



3 different workflows to choose from!

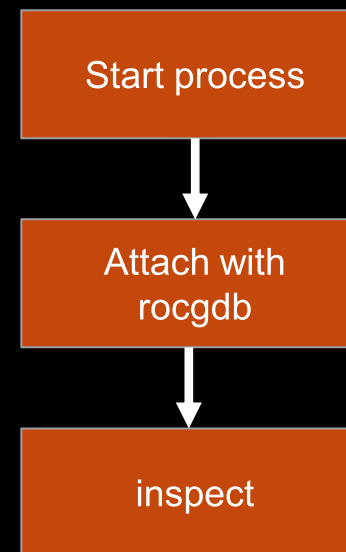
If interested in stepping into GPU code you should avoid gdbserver

Pytorch example app – MNIST distributed learning – Rocgdb

- Examples:
 - PyTorch 1.13 (ROCM 5.2 build)
 - Hanging in collective (RCCL):

```
for epoch in range(3):  
    ...  
    if rank == 6 and epoch == 1:  
        while True:  
            time.sleep(10)
```

Rank 6 will hang after first epoch!



Pytorch example app – MNIST distributed learning – Rocgdb

- After we see the code hanging we need to investigate it by connecting to the relevant processes.
- Starting session in specific nodes to attach from

- For first node of allocation:

```

srun --interactive \
--pty \
/bin/bash

```

- For other nodes nodes (GPU's won't be visible):

```

srun --pty \
--jobid <jobid> \
-w <target_node> \
--mem=0 \
--oversubscribe \
--interactive \
-n 1 -c 56 \
--gpus-per-task=0 \
/usr/bin/bash

```

This is the rank causing the hang
(rank 6 runs on device 6)!

```

$queue --me
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
6920761 dev-g inter samantao R 1:02:38 2 nid[005024-005025]

```

```

$srun --jobid 6920761 --interactive --pty /bin/bash

```

```

user@nid005024$ rocm-smi --showpidgpus

```

```

===== ROCm System Management Interface =====
===== GPUs Indexed by PID =====
PID 6235 is using 1 DRM device(s):
6
PID 6253 is using 1 DRM device(s):
0
PID 6241 is using 1 DRM device(s):
5
PID 6238 is using 1 DRM device(s):
7
PID 6236 is using 1 DRM device(s):
2
PID 6240 is using 1 DRM device(s):
4
PID 6239 is using 1 DRM device(s):
3
PID 6237 is using 1 DRM device(s):
1
=====
===== End of ROCm SMI Log =====

```

My job that is hanging

Connect to the first node in that job allocation.

Query which process is running in which GPU



Pytorch example app – MNIST distributed learning – Rocgdb

- Inspecting rank 1 (hanging waiting for rank 6)

```

$rocgdb attach 6237
GNU gdb (rocm-rel-5.2-109) 11.2
...
Attaching to process 6237
[New LWP 6263]
...
[New LWP 6804]
...
(gdb) info th
Id Target Id Frame
* 1 Thread 0x1489452cc740 (LWP 6237) "python" 0x00001489453dd047 in select () from
/lib64/libc.so.6
...
22 AMDGPU Wave 2:8:?:2 (?,?,?)/1 "python" 0x000014868e215320 in void (anonymous
namespace)::runRing<float, FuncSum<float>, Protocol (ncclWorkElem*) ()
from file:///opt/rocm/lib/librccl.so
...

```

Attach to the process

GDB is attaching...

GDB is registering threads of the attached process

Query threads

CPU thread that was interrupted

GPU thread showing that is hanging in a collective (RCCL)

Dispatch information incomplete as we attached after dispatch was done



Backup - Wave details

agent-id:queue-id:dispatch-num:wave-id (work-group-x,work-group-y,work-group-z)/work-group-thread-index

```
(gdb) i th
```

Id	Target Id	Frame
1	Thread 0x7ffff7fe6e80 (LWP 16912) "saxpy"	0x00007ffffe0fc4c0 in rocr::core::InterruptSignal: t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
2	Thread 0x7ffffd428700 (LWP 16916) "saxpy"	0x00007ffff5e1972b in ioctl () from /lib64/libc.so
4	Thread 0x7ffffecaff700 (LWP 16938) "saxpy"	0x00007ffffe0fc4af in rocr::core::InterruptSignal: t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
* 5	AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"	saxpy (n=256, x=0x7ffffec700000, incx=1, y=0x7ffffec700000)
6	AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"	saxpy (n=256, x=0x7ffffec700000, incx=1, y=0x7ffffec700000)
7	AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"	saxpy (n=256, x=0x7ffffec700000, incx=1, y=0x7ffffec700000)
8	AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"	saxpy (n=256, x=0x7ffffec700000, incx=1, y=0x7ffffec700000)

agent (GPU) ID
(HSA) queue ID
dispatch number
wave ID

workgroup
(x, y, z)

wave ID
(within group)

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

© 2022 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, Radeon, Radeon Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

AWS is a trademark of Amazon.com, Inc. or its affiliates in the United States and/or other countries

Questions?

AMD 