Some more tips & tricks

Samuel Antao

LUMI pre-hackathon training
Oct. 8th, 2024

SINGULARITY CONTAINERS IN LUMI

- Control better the AI framework's 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/
  - Pytorch. Tensorflow, JAX, CuPy, MPI4Py

- 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

SINGULARITY CONTAINERS IN LUMI

```
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.

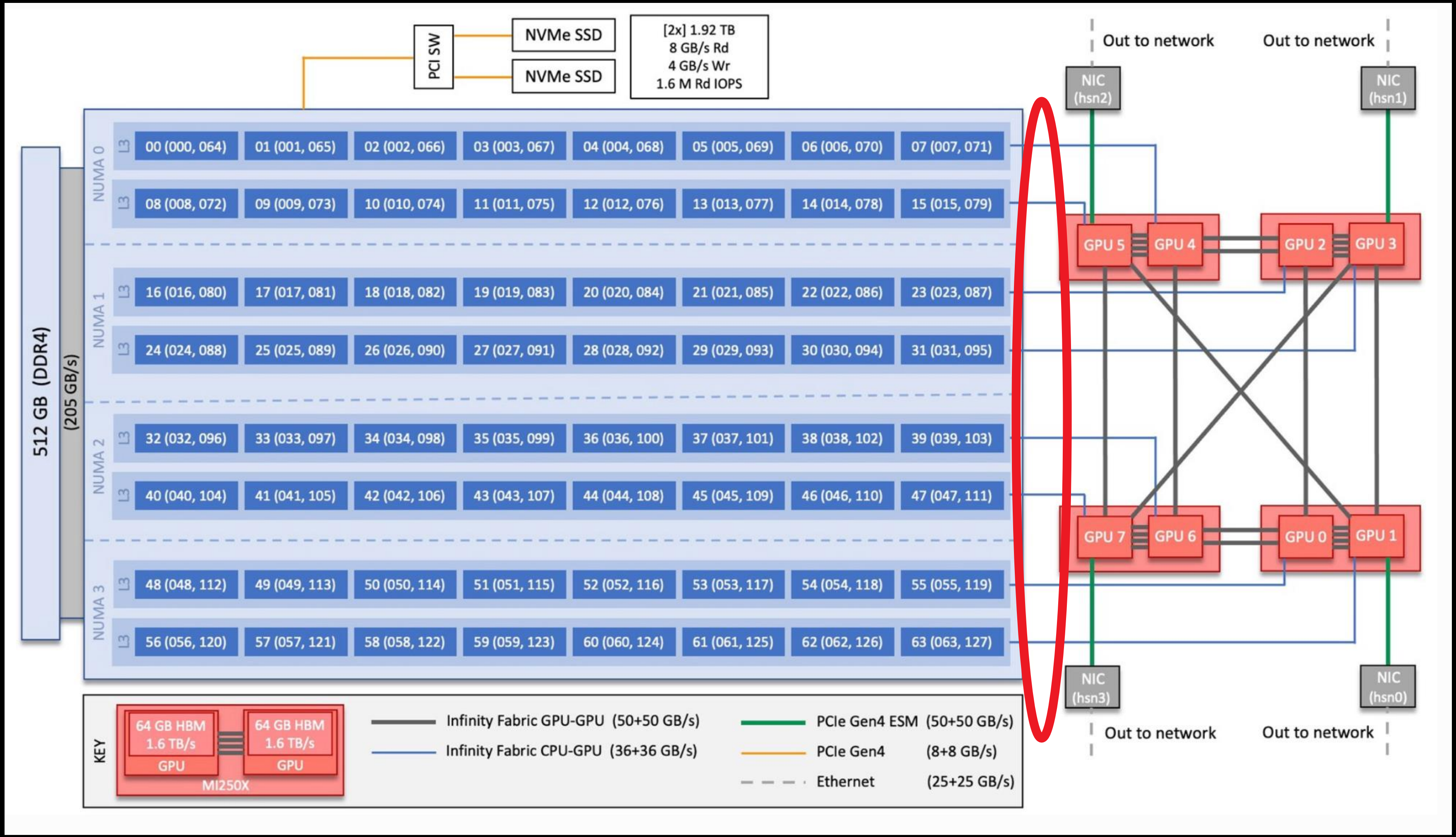
Application Python command

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

[Public]

Checking GPU-CPU affinity

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



AMD
together we advance_

Testing affinity on multiple nodes

- 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
```

Careful! Allocations do not follow GPU ranking!!

AMD together we advance_

Testing affinity on multiple nodes

Interpreted across nodes using a round-robin approach

- Check what SLURM is giving us:

```
srun -N 2 -n 16 --gpus 16 \
 --cpu-bind=mask_cpu:0xfe000000000000,0xfe0000000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000 \
 bash -c 'echo "$SLURM_PROCID -- GPUS $ROCR_VISIBLE_DEVICES -- $(taskset -p $$)"' \
 | sort -n -k1
```

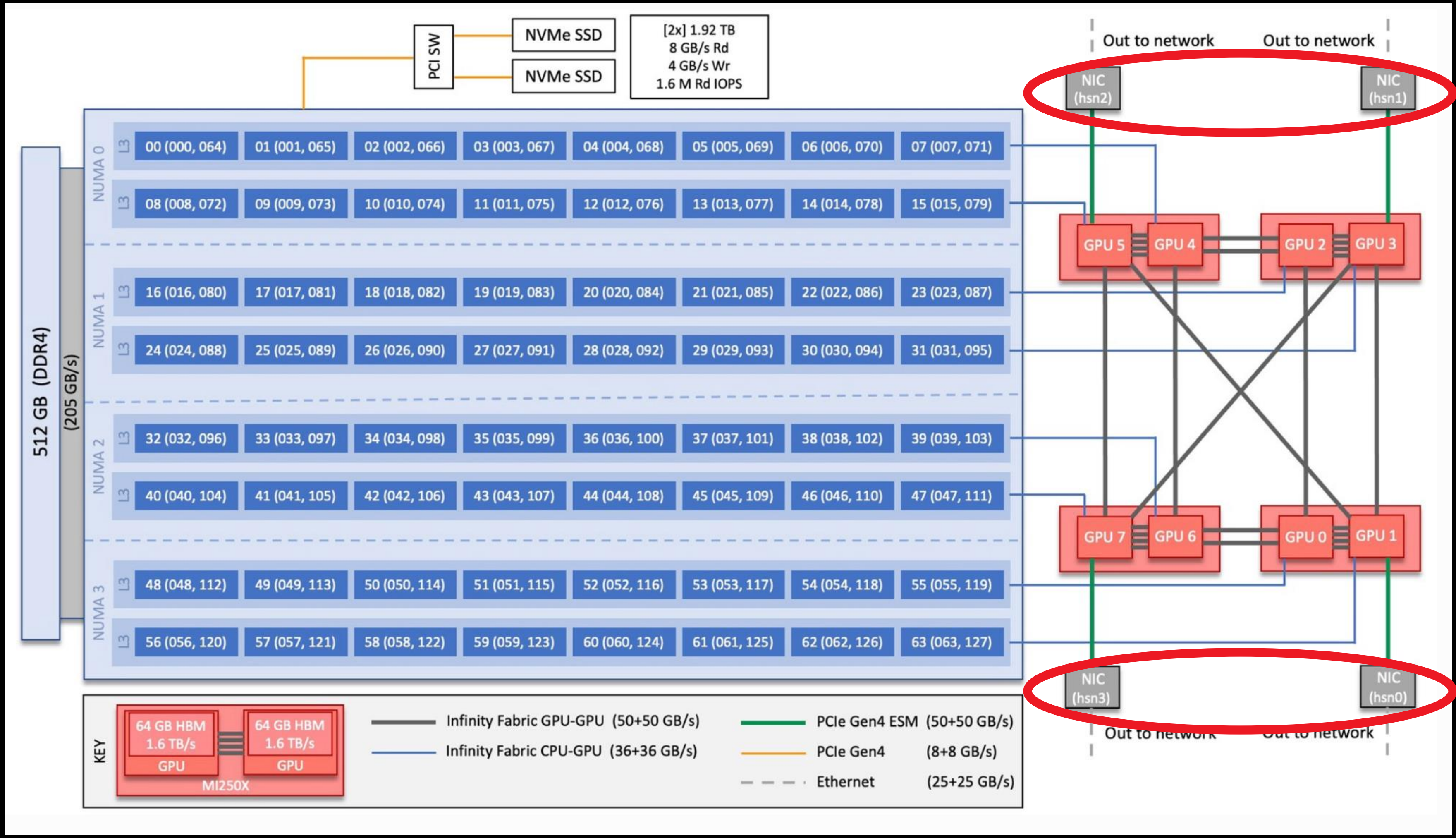
```
0 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13819's current affinity mask: fe000000000000
1 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13820's current affinity mask: fe0000000000000000
2 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13821's current affinity mask: fe0000
3 -- GPUS 0,1,2,3,4,5,6,7 -- pid 13822's current affinity mask: 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: fe0000000000000000
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
```

Consider add an affinity check in your job scripts!

AMD
together we advance_

Checking GPU and NIC connection

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



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

- 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
```

Node has interfaces other than Slingshot

These are the correct ones.

```
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
```

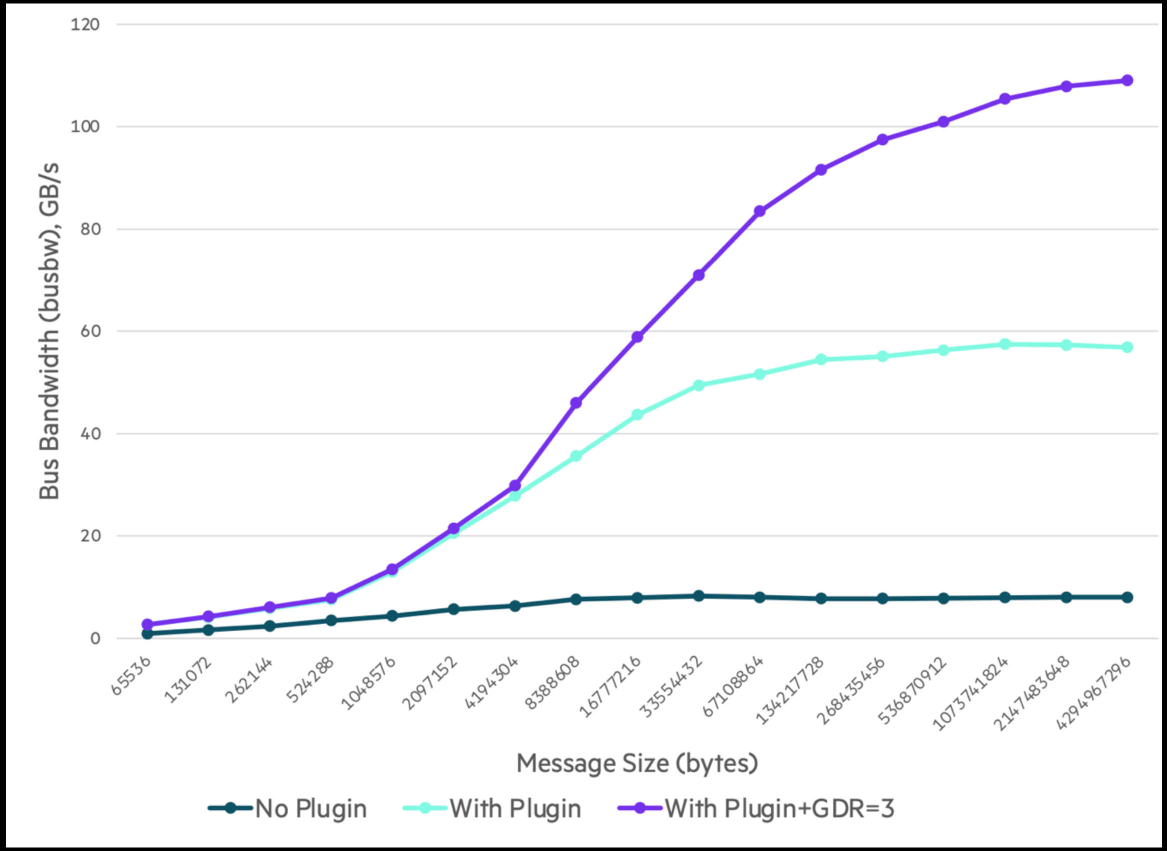
Point RCCL to use all 4 high-speed interfaces. It will know how to bind them based on the node topology.

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_

Where's the master???

- Ranks need to know where the master ranks is:

```
export MASTER_ADDR=$(hostname)
export MASTER_PORT=29500
```

- When using multiple nodes this is not good enough.
- We can leverage SLURM tools to query what the first node of an allocation is:

```
export MASTER_ADDR=$(scontrol show hostname "$SLURM_NODELIST" | head -n1)
export MASTER_PORT=29500
```

- There is no SLURM tools inside the containers:

```
srun singularity exec mycontainer.sif \
bash -c 'MASTER_ADDR=$(scontrol show hostname "$SLURM_NODELIST" | head -n1) ./myapp'
```

```
MASTER_ADDR=$(scontrol show hostname "$SLURM_NODELIST" | head -n1)
srun singularity exec mycontainer.sif \
bash -c './myapp'
```

Putting it all together

- 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
export NCCL_NET_GDR_LEVEL=PHB

# 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
python -u ./myapp
```

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.

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

AMD
together we advance_

Monitoring activity with multiple nodes

- rocm-smi can still be used to understand GPU activity.
- Using SLURM to access nodes other than the first one in the allocation can be challenged.
- You can chose to forward the relevant monitoring information to access from the login node.
- Pipe information to a port of your choosing in your launching script :

```
srun -N 2 -n 2 bash -c 'watch -n1 rocm-smi | nc -l 0.0.0.0 56789'
```

- Access the information from the login node:

```
nc nid007974 56789
```

```
======================= ROCm System Management Interface =======================
================================= Concise Info =================================
GPU  Temp   AvgPwr  SCLK    MCLK     Fan  Perf    PwrCap  VRAM%  GPU%
0    46.0c  92.0W   800Mhz  1600Mhz  0%   manual  500.0W    0%   0%
1    52.0c  N/A     800Mhz  1600Mhz  0%   manual  0.0W      0%   0%
```


AMD
together we advance_

Monitoring activity with multiple nodes - profiling

- Profiling and logging can and (most of the time) should be target at specific ranks.
  - Overhead
  - Cluttered information

- Leverage the SLURM environment to tailor the application instantiation to activate profile or logging.

```
pcmd=''
if [ $SLURM_PROCID -eq 2 ] then
  pcmd='rocprof --hip-trace --stats’
fi
$pcmd ./myapp
```

- If profiling with more than one rank makes sure to define rank-specific output files to avoid corruption.

```
rocprof --hip-trace --stats -o myprofile-$SLURM_PROCID.csv ./myapp
```


AMD
together we advance_

Logging from the environment

- HIP runtime and GPU dispatch information can be logged with AMD_LOG_LEVEL=4

Number of blocks and threads of the dispatch

```
:3:hip_module.cpp        :662 : 117659918626 us: 8088 : [tid:0x14b2015e9700]
    hipLaunchKernel ( 0x14b5ec183ed0, {32768,1,1}, {512,1,1}, 0x14b2015e71b0, 0, stream:<null> )
...
:3:rocvirtual.cpp        :786 : 117659918634 us: 8088 : [tid:0x14b2015e9700] Arg0:   = val:16777216
:3:rocvirtual.cpp        :786 : 117659918636 us: 8088 : [tid:0x14b2015e9700] Arg1:   = val:22689590804480

... ShaderName : _ZN2at6native6legacy18elementwise_kernelILi512ELi1EZNS0_15gpu_kernel_implIZZZNS0_23direct_copy_kernel

:3:hip_module.cpp        :663 : 117659918649 us: 8088 : [tid:0x14b2015e9700] hipLaunchKernel: Returned hipSuccess :
```

Arguments

Kernel mangled name

Return error.

AMD together we advance_

[Public]

Leveraging framework profiler infrastructure

- 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

AMD
together we advance_

Disclaimer and Attributions

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18