Extreme scale AI training on LUMI

Samuel Antao

LUMI AI Workshop
Ostrava, Czechia, Nov. 26-27th, 2024

Controlling device visibility with multiple nodes

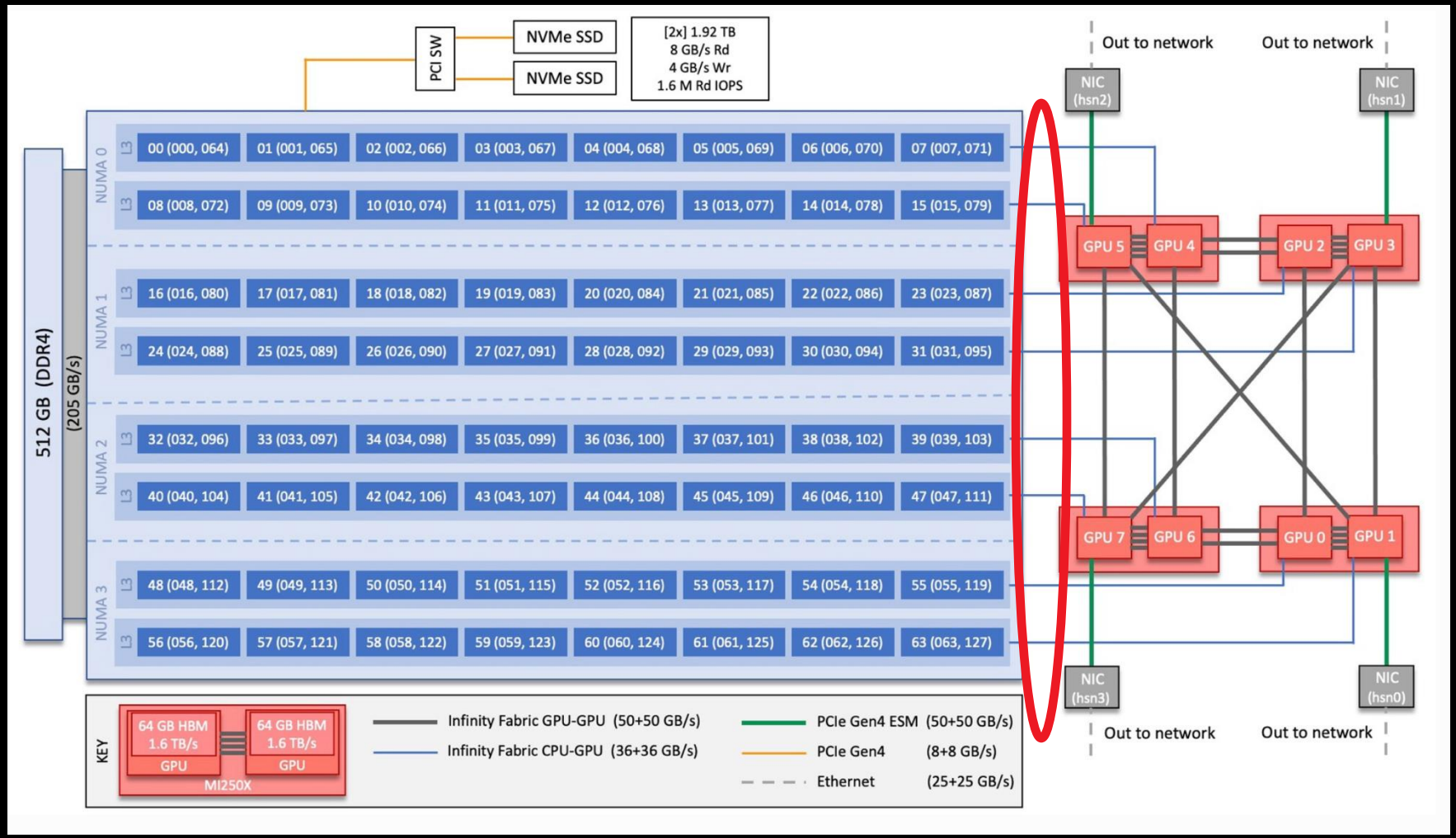
- Rank and global rank now mean different things!
- Controlling visibility
  - HIP_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())'
  - ROCR_VISIBLE_DEVICES=0,1,2,3 python -c 'import torch; print(torch.cuda.device_count())’
  - SLURM sets ROCR_VISIBLE_DEVICES
  - Implications 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!!!

Checking GPU-CPU affinity

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



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!!

Testing affinity on multiple nodes

- Check what SLURM is giving us:

Interpreted across nodes using a round-robin approach

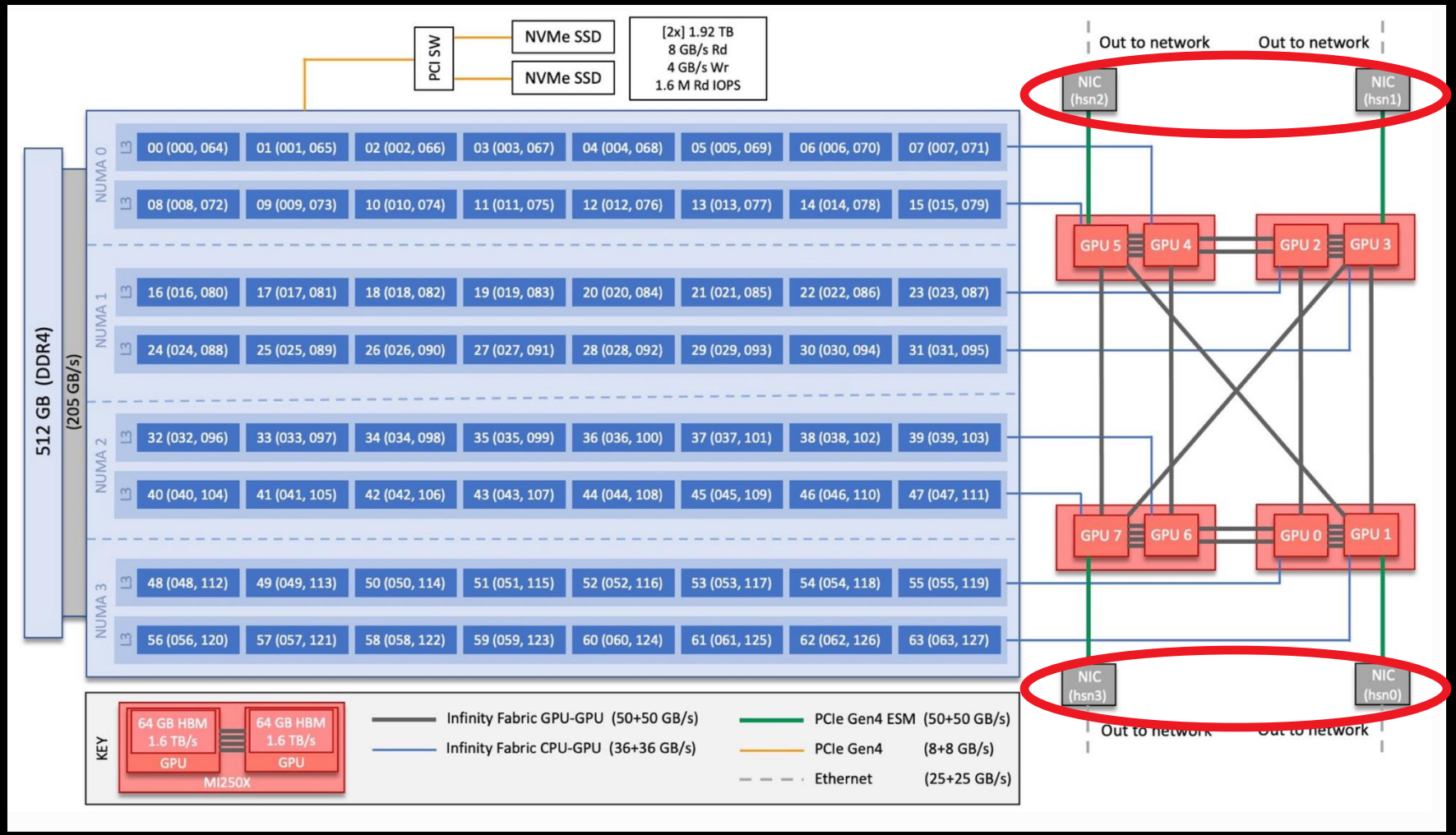
```
srun -N 2 -n 16 --gpus 16 \
 --cpu-bind=mask_cpu:0xfe000000000000,0xfe00000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000 \
 bash -c 'echo "$SLURM_PROCID -- GPUS $ROCR_VISIBLE_DEVICES -- $(taskset -p $$)"' \
 | sort -n -k1
```

```
 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
```

Consider add an affinity check in your job scripts!

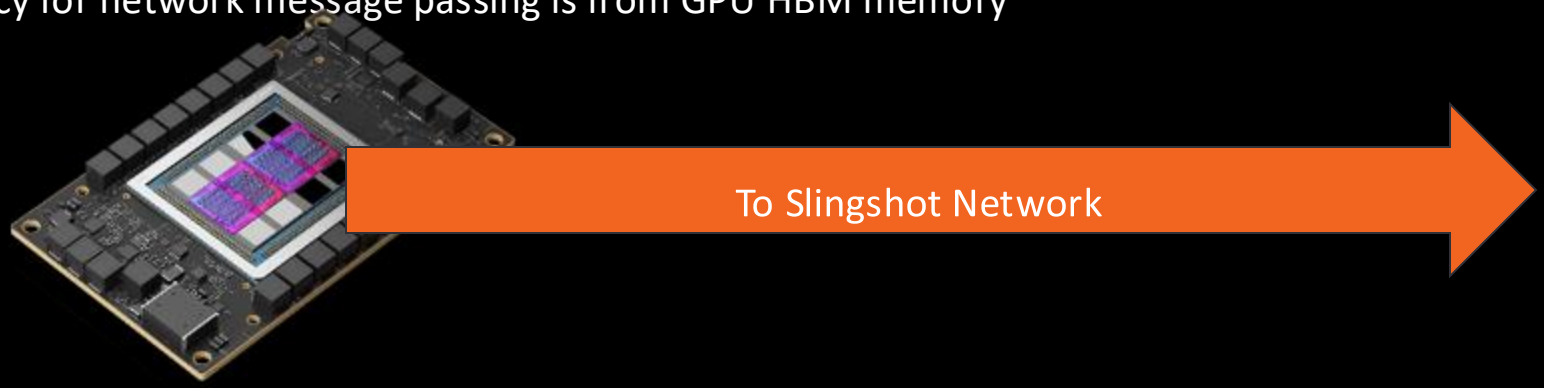
Checking GPU and NIC connection

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



Comms are important!

- LUMI, Frontier (and others) directly attaches AMD Instinct™ MI250x Accelerator to the Slingshot Network
  - Minimize the role of the CPU in the control path
  - Lowest latency for network message passing is from GPU HBM memory



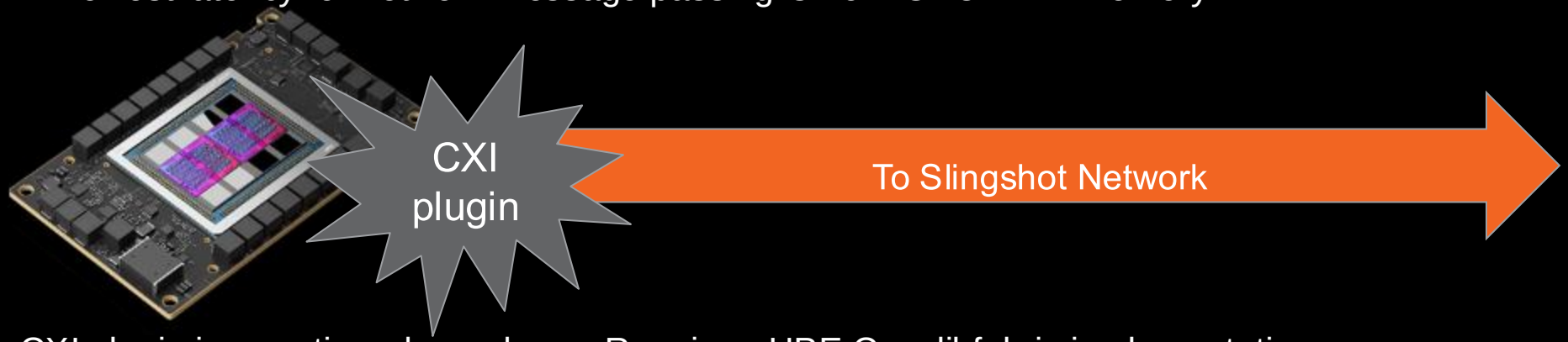
- For example,

```
hipMalloc(&device_buffer, Msize);
init_data<< >>(device_buffer, Msize);  // launch GPU kernel
hipDeviceSynchronize();
MPI_Sendrecv(&device_buffer, Msize, MPI_FLOAT, … , MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

- Requires: HPE Cray MPICH

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
```

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

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:

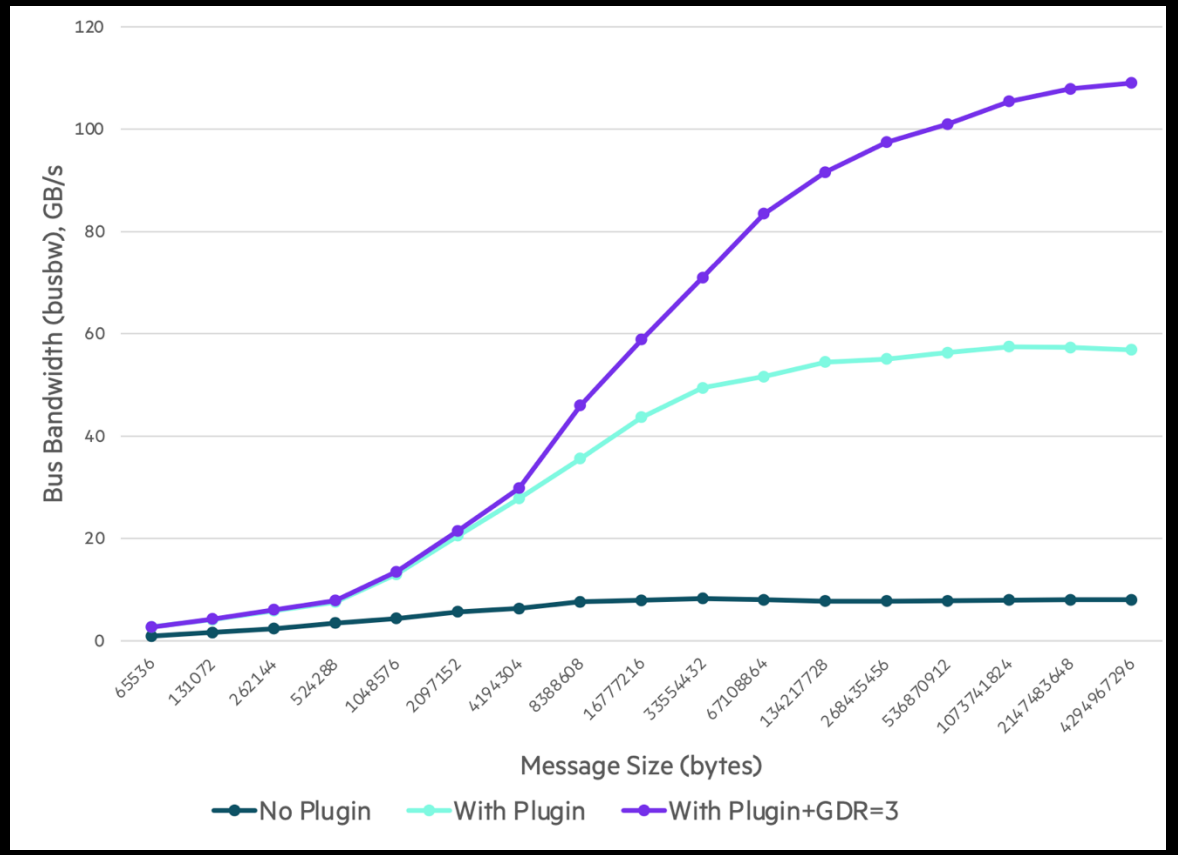
```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
```

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

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!**



MIOpen configuration

- MIOpen is a library for high-optimized machine learning primitives
- Used on many models – not in our LLM example though
- It uses caches to enable just-in-time compilation organized as SQLite databases
- File system doesn’t deal well with SQLite locks when many processes are trying to access it.
- Solution? Setup individual caches for groups of ranks – we recommend per node:

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

- Want to check MIOpen activity – setup the following environment variable:

  export MIOPEN_ENABLE_LOGGING=1

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

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%
```

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.

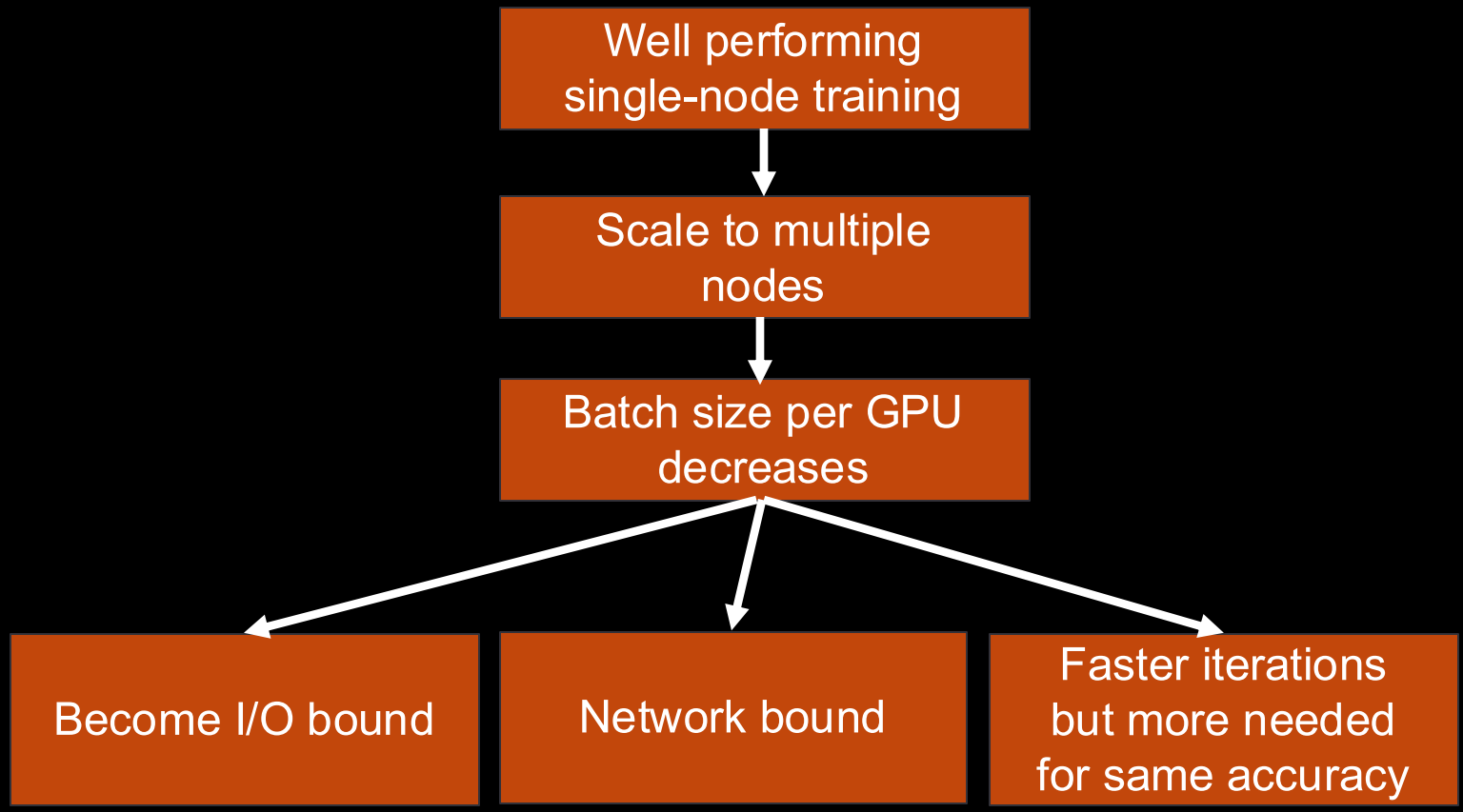
```
pcmd=''
if [ $SLURM_PROCID -eq 2 ] then
  pcmd='rocprof --hip-trace --stats’
fi
$pcmd ./myapp
```

- 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
```

Monitoring is your friend

- Why would I want to scale my model?
  - Train faster – strong-scaling
  - Train bigger – weak-scaling
  - My model doesn't fit in just a few GPUs
- How far can I go?
  - Depends on your model
  - Scaling can change the bottlenecks
  - Scaling can change convergence
- Monitor the regime in which your operating the GPUs at all times!



- You'll always be bound by some type of communication at some point!!!

Other ways to express parallelism - FSDP

- We talked mostly about Distribute Data Parallel (DDP) applications – there are others!
- Fully Sharded Data Parallel is another option.
  - Create shards out of the neural net model more likely to be activated together
  - Try keep less state in the GPU – could support larger models with less GPUs
  - More complexities in configuring the different knobs. Depending on the tunning may require more or less changes to your code.
  - https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/
- Using FSDP requires wrapping your model into the relevant FSDP object.

```
wrapper_kwargs = Dict(cpu_offload=CPUOffload(offload_params=True))
with enable_wrap(wrapper_cls=FullyShardedDataParallel, **wrapper_kwargs):
 fsdp_model = wrap(model())
```

Your original model

- Some tools to control the wrapping in less intrusive ways have been created - ***accelerate***.
  - Enabling FSDP on transformers: https://huggingface.co/docs/transformers

Other ways to express parallelism - Horovod

- Horovod is a framework to enable distributed deep-learning training with TensorFlow, Keras, PyTorch, and Apache MXNet. The goal of Horovod is to make distributed deep learning fast and easy to use.

Step 1: configure ROCm details

```
# Configure for ROCm
export HOROVOD_WITHOUT_MXNET=1
export HOROVOD_WITHOUT_GLOO=1
export HOROVOD_GPU=ROCM
export HOROVOD_ROCM_HOME=$ROCM_PATH
export HOROVOD_GPU_OPERATIONS=NCCL
export HOROVOD_CPU_OPERATIONS=MPI
export HOROVOD_WITH_MPI=1
export HOROVOD_ROCM_PATH=$ROCM_PATH
export HOROVOD_RCCL_HOME=$ROCM_PATH/rccl
export RCCL_INCLUDE_DIRS=$ROCM_PATH/rccl/include
export HOROVOD_RCCL_LIB=$ROCM_PATH/rccl/lib
export HCC_AMDGPU_TARGET=gfx90a
export CMAKE_PREFIX_PATH=$MPICH_PATH
```

Step 2: configure for your favorite framework details

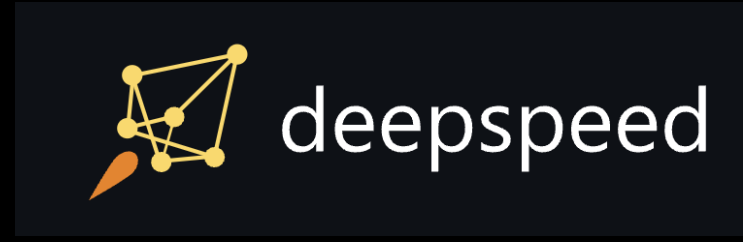
```
# Configure for TensorFlow
export HOROVOD_WITH_TENSORFLOW=1
export HOROVOD_WITHOUT_PYTORCH=1

# Configure for PyTorch
export HOROVOD_WITHOUT_TENSORFLOW=1
export HOROVOD_WITH_PYTORCH=1
```

Horovod needs MPI at launch

Step 3: install

```
# Install
pip install --no-cache-dir --force-reinstall --verbose horovod==$HOROVOD_VERSION
```

Other ways to express parallelism - DeepSpeed



- DeepSpeed is a framework to optimize distributed deep-learning training and inference

```
DS_BUILD_AIO=0 \
DS_BUILD_CCL_COMM=1 \
DS_BUILD_CPU_ADAM=0 \
DS_BUILD_CPU_LION=0 \
DS_BUILD_EVOFORMER_ATTN=0 \
DS_BUILD_FUSED_ADAM=1 \
DS_BUILD_FUSED_LION=1 \
DS_BUILD_CPU_ADAGRAD=0 \
DS_BUILD_FUSED_LAMB=1 \
DS_BUILD_QUANTIZER=0 \
DS_BUILD_RANDOM_LTD=0 \
DS_BUILD_SPARSE_ATTN=0 \
DS_BUILD_TRANSFORMER=0 \
DS_BUILD_TRANSFORMER_INFERENCE=0 \
DS_BUILD_STOCHASTIC_TRANSFORMER=1 \
pip install deepspeed==0.14.0 \
--global-option="build_ext" --global-option="-j32"

ds_report
```

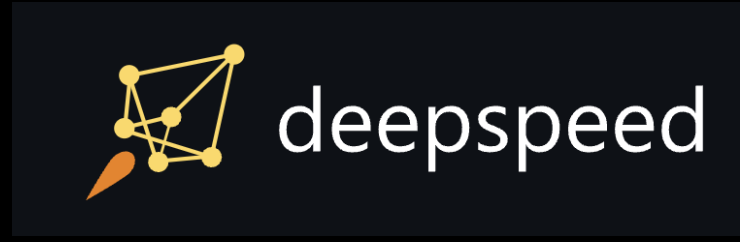
Select all the optimizations not all are enabled for GPUs.

Allow multiple process builds.

Utility to report supported capabilities.

```
--------------------------------------------------
op name ..... installed .. compatible
--------------------------------------------------
async_io ..... [NO] ..... [NO]
fused_adam ..... [YES] ..... [OKAY]
cpu_adam ..... [NO] ..... [OKAY]
cpu_adagrad ..... [NO] ..... [OKAY]
cpu_lion ..... [NO] ..... [OKAY]
evoformer_attn ..... [NO] ..... [NO]
fused_lamb ..... [YES] ..... [OKAY]
fused_lion ..... [YES] ..... [OKAY]
inference_core_ops ..... [NO] ..... [OKAY]
cutlass_ops ..... [NO] ..... [OKAY]
transformer_inference .. [NO] ..... [OKAY]
quantizer ..... [NO] ..... [OKAY]
ragged_device_ops ..... [NO] ..... [OKAY]
ragged_ops ..... [NO] ..... [OKAY]
random_ltd ..... [NO] ..... [OKAY]
sparse_attn ..... [NO] ..... [NO]
spatial_inference ..... [NO] ..... [OKAY]
transformer ..... [NO] ..... [OKAY]
stochastic_transformer . [YES] ..... [OKAY]
--------------------------------------------------
```

Other ways to express parallelism - DeepSpeed



- Again wrapping your model in the relevant object is the way to go!

```
import deepspeed

deepspeed.init_distributed()

model, optimizer, _, _ = deepspeed.initialize(
    model = model,
    optimizer = optimizer, #e.g. SGD
    args = args,
    dist_init_required=True
)
```

Original model

Original optimizers

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