AI Workflow Examples

Presenter: Jose Noudohouenou
AMD Datacenter Solutions Group
Oct 13-16, 2025

AMD
together we advance_

AGENDA

1. Virtual Environment
2. Apptainer
3. Podman
4. Modules (System Installed)

MNIST handwritten digits database

- The MNIST example uses machine learning to recognize handwritten digits from a large database at the National Institute of Standards and Technology. A sample of the handwritten data is shown to the right.
- The PyTorch examples set contains an MNIST example that uses a Convolutional Neural Network (ConvNets) to train an image recognition model.

- For the single GPU runs, we add a line to confirm that the code is running on the GPU.

```
    print(f"Using device: {device}")' main.py`
```

By Suvanjanprasai - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=156115980

MNIST python code

- Retrieve the example code in the virtual environment and modify it as before

```
git clone --depth=1 https://github.com/pytorch/examples.git pytorch_examples
cd pytorch_examples/mnist
```

- Add a line to print out the device that is found

```
101     if use_accel:
102         device = torch.accelerator.current_accelerator()
103     else:
104         device = torch.device("cpu")
105     print(f"Using device: {device}")
```

- Line editor version for batch jobs

```
sed -i -e '/device = torch.device("cpu")/a\    print(f"Using device: {device}")' main.py
```

AAC6 – Total 25 GPUs – your local system may be different

AAC6 is all pre-production parts

Login or Control Node

AAC6:
pl1vm1mi300ctln01

Slurm

salloc
srun
sbatch

Queue 1CN192C4G1H_MI300A_Ubuntu22

ppac-pl1-s24-16
ppac-pl1-s24-26
ppac-pl1-s24-30
ppac-pl1-s24-35

4 MI300A
192 CPUs
512 GB memory

ppac-pl1-s24-16

Queue 1CN48C1G1H_MI300A_Ubuntu22

sh5-pl1-s12-09
sh5-pl1-s12-12
sh5-pl1-s12-15
sh5-pl1-s12-33

1 MI300A
48 CPUs
128 GB memory

sh5-pl1-s12-36

Virtual Environment

- A python virtual environment will isolate our run from the rest of the system
- At the same time, it creates a wholly contained environment, making it more robust
- Lastly, it makes the cleanup of the job easier and more complete

- We'll show a few versions of running in a virtual environment
  - Running on a workstation with a GPU
  - Using an interactive Slurm allocation
  - Submitting a batch version

Virtual Environment on a Workstation with GPU

- Change to the directory with the mnist example
```
cd ~/pytorch_examples/mnist
```
- Create the virtual environment
```
python3 -m venv rocm-pytorch
source rocm-pytorch/bin/activate
```
- Install the ROCm version of PyTorch – specifically torch and torchvision. Time it.
```
time pip3 install torch torchvision --index-url https://download.pytorch.org/whl/rocm6.4
```
- Verify that torch is working and that there is a GPU
```
python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'
```
- Run the image classification with a convolutional neural network. Time it.
```
time python3 main.py
```
- Cleanup
```
deactivate
rm -rf rocm-pytorch
```

Virtual Environment Interative Slurm or PBS allocation

- Get an allocation with a GPU to run this example.
  ```
  salloc –ntasks=1 --gpus=1 --time=01:00:00
  ```
- or
  ```
  srun --ntasks=1 –gpus=1 –pty -- bash
  ```
- Same steps interactively as for the workstation w/GPU

- Exit the GPU allocation
  ```
  exit
  ```

- PBS version: the `–I` specifies interactive
  ```
  qsub –I –l select=1:ncpus=1:ngpus=1 walltime=01:00:00
  ```
- To check your jobs
  ```
  squeue –u <username>          (SLURM)
  qstat –u <username>           (PBS)
  ```
- To check all jobs that are running
  ```
  squeue                        (SLURM)
  qstat                         (PBS)
  ```
- To get the information about the queues
  ```
  sinfo                         (SLURM)
  qstat –q                      (PBS)
  ```
- To cancel a job
  ```
  scancel <job number>          (SLURM)
  qdel <job number>             (PBS)
  ```

Virtual Environment in a Slurm batch job

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --output=pytorch_mnist_venv.out
#SBATCH --error=pytorch_mnist_venv.out

python3 -m venv rocm-pytorch
cd rocm-pytorch
source bin/activate
echo "Starting pytorch install"
time pip3 install torch torchvision --index-url https://download.pytorch.org/whl/rocm6.4

python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'

git clone --depth=1 https://github.com/pytorch/examples.git pytorch_examples
cd pytorch_examples/mnist
sed -i -e '/device = torch.device("cpu")/a\    print(f"Using device: {device}")' main.py

pip3 install -r requirements.txt

echo "Starting mnist"
time python3 main.py
cd ../../..

deactivate
rm -rf rocm-pytorch

echo "Finished"
```

Submit with
`sbatch pytorch_mnist_venv.batch`

Check job is running with "`squeue`"

Watch output with
`tail –f pytorch_mnist_venv.out`

Virtual Environment in a Slurm batch job

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --output=pytorch_mnist_venv.out
#SBATCH --error=pytorch_mnist_venv.out

python3 -m venv rocm-pytorch
cd rocm-pytorch
source bin/activate
echo "Starting pytorch install"
time pip3 install torch torchvision --index-url https://download.pytorch.org/whl/rocm6.4

python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'

git clone --depth=1 https://github.com/pytorch/examples.git pytorch_examples
cd pytorch_examples/mnist
sed -i -e '/device = torch.device("cpu")/a\    print(f"Using device: {device}")' main.py

pip3 install -r requirements.txt

echo "Starting mnist"
time python3 main.py
cd ../../..

deactivate
rm -rf rocm-pytorch

echo "Finished"
```

```
#PBS -l select=1:mpiprocs=1:ompthreads=1:node_type=mi300a
#PBS -l walltime=01:00:00
```

(consult also the wiki of the system you use for site specific and version details)

Submit with

`sbatch pytorch_mnist_venv.batch`

In PBS: qsub `pytorch_mnist_venv.batch`

Check job is running with "`squeue`"

Watch output with

`tail -f pytorch_mnist_venv.out`

Virtual Environment

- Note that the importing of torch takes a large fraction of the time (~7min) versus training (~2min)
  - We could keep the environment rather than deleting it after the run

- Larger training runs will dwarf the setup time.
- Staging and reading of data sets will become more important.

Apptainer

- Download the image for the apptainer run prior to job

```
salloc --exclusive  -p 1CN48C1G1H_MI300A_Ubuntu22
apptainer pull rocm-pytorch.sif docker://rocm/pytorch:rocm6.4.3_ubuntu24.04_py3.12_pytorch_release_2.6.0
exit
```

  - We use the sh5 nodes to avoid tying up GPUs. The SH5 nodes have a single GPU. If you have a CPU only node, that would be even better.
  - Multiple cores (CPUs) can speed up the download and conversion process, but you are still limited by network bandwidth
  - Pulling an SIF file does not benefit from having more cores. It only helps when converting Docker files.
  - Filesystem speeds can also limit performance
- To clean up afterwards

```
rm –f rocm-pytorch.sif
apptainer cache clean
```

- Images are cached for future use on any node
  - Even restoring the SIF file from a cached image takes significant time
  - Reuse the rocm-pytorch.sif file to avoid the start-up time
  - SIF files are large, so don't download them all over your filesystem and be sure to clean up afterwards

Apptainer example using an interactive Slurm allocation

- Get an allocation with a single GPU and start up an Apptainer shell

```
salloc --ntasks 1 --gpus=1 --time=01:00:00
apptainer shell --rocm ~/rocm-pytorch.sif
```

- Check that torch is available and that there is a usable GPU

```
python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'
```

- Prepare to run the example

```
cd pytorch_examples/mnist
pip3 install --user -r requirements.txt
```

- Run the python code and time it.

```
time python3 main.py
```

- Exit the shell and then the allocation – two exits!

```
exit
exit
```

Apptainer example using a Slurm batch job

- Slurm batch script pytorch_mnist_apptainer.batch

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --output=pytorch_mnist_apptainer.out
#SBATCH --error=pytorch_mnist_apptainer.out

mkdir -p .torch

cd $HOME/pytorch_examples/mnist

apptainer exec --rocm \
  --bind "$PWD:/workspace" -W /workspace \
  --env TORCH_HOME=/workspace/.torch \
  ~/rocm-pytorch.sif  \
pwd
python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'

pip3 install --user -r requirements.txt
time python3 main.py
```

Submit with

```
sbatch pytorch_mnist_apptainer.batch
```

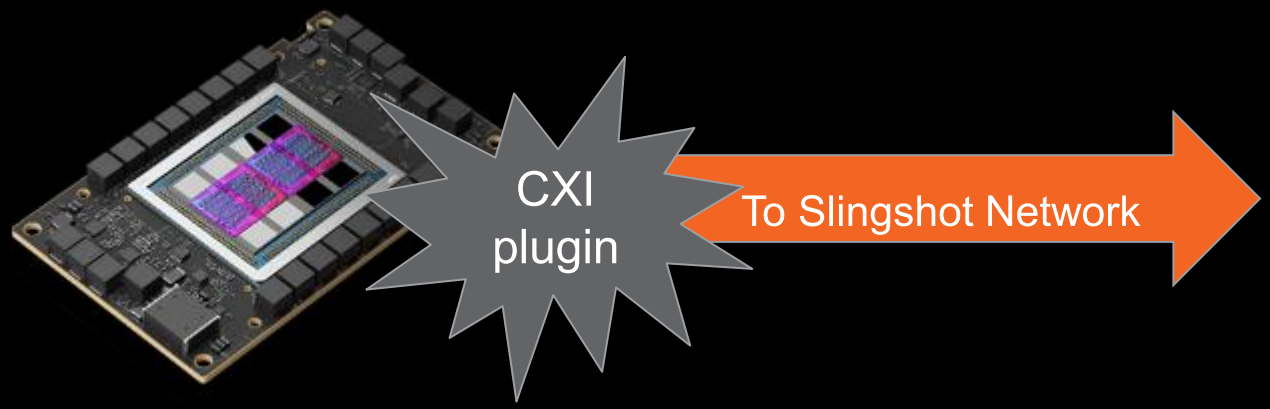
Check job is running with "squeue"

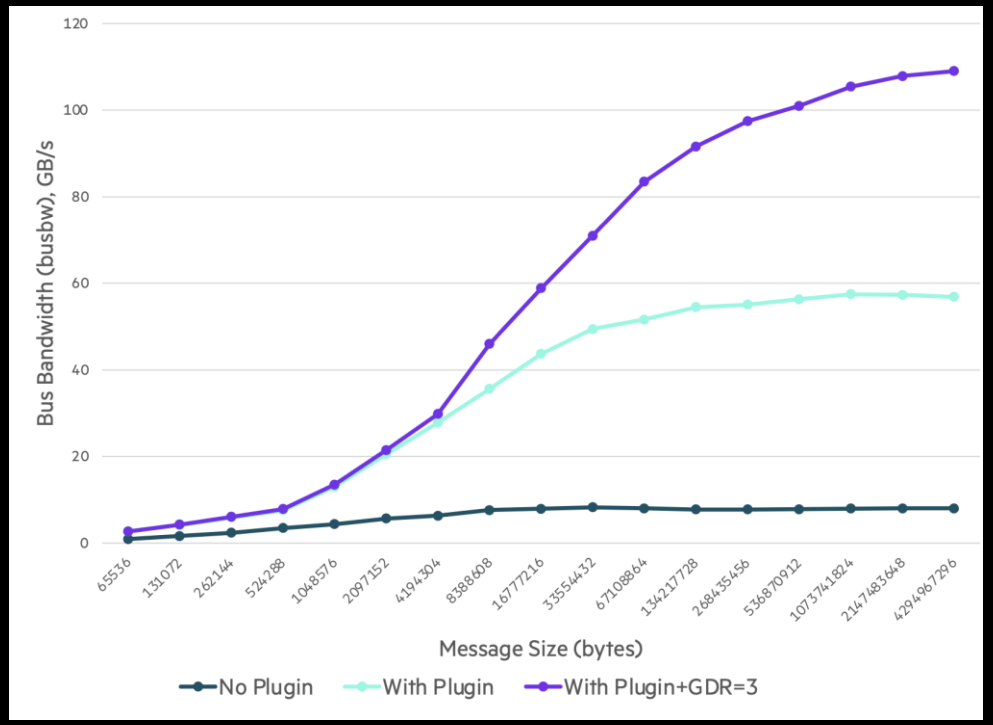
Watch output with

```
tail –f pytorch_mnist_apptainer.out
```

Apptainer – system-specific requirements

- Container are convenient way to convey system-specific functionality to users and facilitate site support
  - E.g. CINES, LUMI, Frontier (and others) directly attaches AMD Instinct™ Accelerators 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

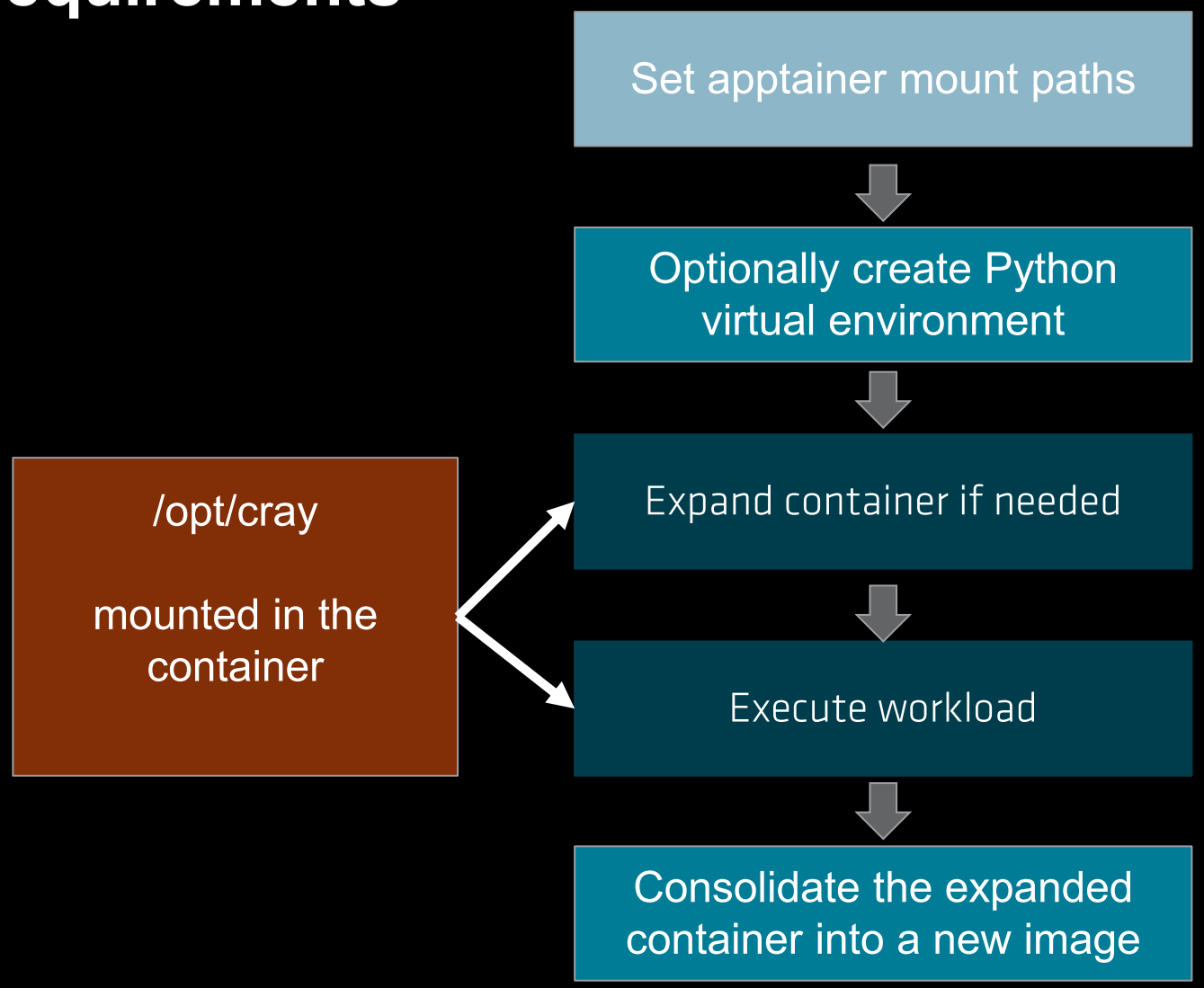




- Requires HPE Cray libfabric implementation (proprietary)
  - Possible to build from recent releases
- Optimized MPI implementations leverage same logic

Apptainer – system-specific requirements

- Utilize base images closest to the host system
  - E.g. SLES 15 for CrayOS
- Docker images converted to apptainer and made available on the file system
- Two options:
  - Use image directly – user needs to mount dependencies
  - Leverage Easybuild/Spack to setup the relevant environment
- Extending the container.
  - Is time consuming to recreate iterate on apptainer container builds
  - Cotainr – sandbox-based extension
    - https://github.com/DeiC-HPC/cotainr
  - Virtual environment on filesystem (created by Easybuild/Spack)
- Consider consolidating the expenanded container into a new image



Apptainer – system-specific requirements - extend

- Extend container python installation with a virtual environment
  - Way faster than having to recreate a new container

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --output=pytorch_extend.out
#SBATCH --error=pytorch_extend.out

apptainer exec --rocm \
  --bind "$(pwd)"
  ~/rocm-pytorch.sif  \
  bash –c  ‘
   python3 –m venv –system-site-packages venv ;
   source venv/bin/activate ;
   pip install transformers’
```

Extend the existing packages

Install extra packages

Submit with

`sbatch pytorch_extend.batch`

Check job is running with "`squeue`"

Watch output with

`tail –f pytorch_extend.out`

Apptainer – system-specific requirements - run

- Containers are privileged way to control

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --output=pytorch_extended_run.out
#SBATCH --error=pytorch_extended.out

apptainer exec --rocm \
  --bind "$(pwd)"  \
  --bind /opt/cray \
  ~/rocm-pytorch.sif  \
  bash –c ‘
   source venv/bin/activate ;
   python3 -c 'import transformers; print(transformers.__version__)'
```

Use proprietary libraries from the host

Use package from virtual environment

Submit with

`sbatch pytorch_extended_run.batch`

Check job is running with "`squeue`"

Watch output with

`tail -f pytorch_extended_run.out`

Podman

- Podman is a docker alternative that provides a rootless environment
- Docker images can be directly used from docker hub
- Podman images are stored in a local directory on each node
  - requires a separate download and image on each node
- Podman is not installed on the login node
- Cleaning up uses similar commands to docker
  - `podman image prune`
  - `podman system prune`
  - `podman system prune --all`

Podman example with interactive Slurm allocation

- Get an interactive Slurm allocation with a GPU

```
salloc --ntasks 1 --gpus=1 --time=01:00:00
```

- And start up an interactive Podman shell

```
podman run -it --network=host --device=/dev/kfd --device=/dev/dri --group-add=video --ipc=host \
  --cap-add=SYS_PTRACE --security-opt seccomp=unconfined  -v $HOME:/workdir -w /workdir \
  docker://rocm/pytorch:rocm6.4.3_ubuntu22.04_py3.10_pytorch_release_2.6.0 bash
```

- Prepare the environment

```
cd pytorch_examples/mnist
pip3 install -r requirements.txt

python3 - << EOF
import torch, platform
print("Torch module location: ",torch)
print("Torch:", torch.__version__, " HIP:", getattr(torch.version, "hip", None))
print("Platform:", platform.platform())
print("torch.cuda.is_available:", torch.cuda.is_available())
print("Device count:", torch.cuda.device_count())
if torch.cuda.is_available():
    print("Device 0:", torch.cuda.get_device_name(0))
EOF
```

- Run the mnist examples

```
time python3 main.py
```

Podman example with Slurm batch job

- Slurm batch script pytorch_mnist_podman.batch

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --output=pytorch_mnist_podman.out
#SBATCH --error=pytorch_mnist_podman.out

cd $HOME/pytorch_examples/mnist

podman run --rm \
  --device=/dev/dri --device=/dev/kfd --network=host --ipc=host \
  --userns=keep-id --group-add=keep-groups --cgroupns=host \
  --cap-add=SYS_PTRACE --security-opt seccomp=unconfined \
  -v $PWD:/workdir -w /workdir \
  docker://rocm/pytorch:rocm6.4.3_ubuntu22.04_py3.10_pytorch_release_2.6.0 \
pip3 install --user -r requirements.txt

python3 - << EOF
import torch, platform
print("Torch module location: ",torch)
print("Torch:", torch.__version__, " HIP:", getattr(torch.version, "hip", None))
print("Platform:", platform.platform())
print("torch.cuda.is_available:", torch.cuda.is_available())
print("Device count:", torch.cuda.device_count())
if torch.cuda.is_available():
    print("Device 0:", torch.cuda.get_device_name(0))
EOF

time python3 main.py
```

Submit with
`sbatch pytorch_mnist_podman.batch`

Check job is running with "`squeue`"

Watch output with
`tail –f pytorch_mnist_podman.out`

Modules – System Installation

Beware that e.g. lustre does not like many small files, that may become a challenge for some AI datasets and framework installations

- For a more typical HPC system installation, we can custom build PyTorch and its friends, place it on the system and provide a module to use it.
  - Can be done with pip installs to the installation directory
  - Or build from source for better performance and leaner installations
    - Just the gfx models needed on the system – gfx90a (MI200 series) and gfx942 (MI300 series)
  - The PyTorch installation on this system includes:
    - torch
    - torchaudio
    - triton
    - transformers
    - sageattention
    - flashattention
- The module prepends the necessary paths to PYTHONPATH for the PyTorch packages. Either clear it of any unneeded and system paths before loading the module or use a virtual environment.

Modules – interactive Slurm

- Get a GPU allocation

```
salloc --ntasks 1 --gpus=1 --time=01:00:00
```

- Load the environment and run a basic check that it is working

```
module load rocm pytorch
cd ~/pytorch_examples/mnist
pip3 install --user -r requirements.txt
python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'
```

- We can run a more detailed system check

```
python3 - << EOF
import torch, platform
print("Torch module location: ",torch)
print("Torch:", torch.__version__, " HIP:", getattr(torch.version, "hip", None))
print("Platform:", platform.platform())
print("torch.cuda.is_available:", torch.cuda.is_available())
print("Device count:", torch.cuda.device_count())
if torch.cuda.is_available():
    print("Device 0:", torch.cuda.get_device_name(0))
EOF
```

- Now run our pytorch application

```
time python3 main.py
```

- Exit the allocation

```
exit
```

Modules – interactive Slurm with a virtual environment

- It is a little cleaner to use a virtual environment as discussed earlier
- Get a GPU allocation

```
salloc --ntasks 1 --gpus=1 --time=01:00:00
```

- Create the virtual environment

```
cd ~/pytorch_examples/mnist
python3 -m venv rocm-pytorch
source rocm-pytorch/bin/activate
pip3 install --user -r requirements.txt
```

- Run the same sequence of checks as before (optional)

- Now run our pytorch application

```
time python3 main.py
```

- Exit the virtual environment and clean up

```
deactivate
rm –rf rocm-pytorch
```

- Exit the allocation

```
exit
```

Modules – submit Slurm batch job with a virtual environment

- The batch file pytorch_mnist_module_venv.batch

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --time=01:00:0
#SBATCH --ntasks=1
#SBATCH --output=pytorch_mnist_module_venv.out
#SBATCH --error=pytorch_mnist_module_venv.out

module load rocm pytorch

cd ~/pytorch_examples/mnist

python3 -m venv rocm-pytorch
source rocm-pytorch/bin/activate
pip3 install -r requirements.txt

... Check environment commands ...

time python3 main.py
deactivate
rm -rf rocm-pytorch
```

- Submit the batch file with sbatch pytorch_mnist_module_venv.batch

Summary

- A variety of methods can be used to run PyTorch applications
- The set up time, network load and disk space requirements varies for each
- Matching versions is easier when building a custom image – modules
- Environment in a pre-build docker or wheel might be unexpected and path sensitive
- Reproducibility requires some effort with each method

Modifications to run on multiple GPUs

- Edit the main.py script in pytorch_examples/mnist and change

```
model = Net().to(device)
```

- to

```
model = Net()
if torch.cuda.device_count() > 1:
    print(f"Using {torch.cuda.device_count()} GPUs!")
    model = nn.DataParallel(model)
model.to(device)
optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
```

- A sed script to change it on the fly for batch jobs

```
sed -i -e '/device = torch.device("cpu")/a\    print(f"Using device: {device}")' main.py
sed -i -e '/model = Net().to(device)/s/.to(device)//' main.py
sed -i -e '/model = Net()/a\    if torch.cuda.device_count() > 1:\
        print(f"Using {torch.cuda.device_count()} GPUs!")\
        model = nn.DataParallel(model)\
    model.to(device)' main.py
```

Virtual Environment batch Slurm script

- Batch script pytorch_mnist_venv_2gpus.batch

```
#!/bin/bash
#SBATCH --gpus=2
#SBATCH --time=01:00:0
#SBATCH --ntasks=2
#SBATCH --output=pytorch_mnist_2gpus.out
#SBATCH --error=pytorch_mnist_2gpus.out

module load rocm pytorch

cd ~/pytorch_examples/mnist
python3 -m venv rocm-pytorch-2gpus
source rocm-pytorch-2gpus/bin/activate
pip3 install torch torchvision --index-url https://download.pytorch.org/whl/rocm6.4

python3 -c 'import torch' 2> /dev/null && echo 'Success' || echo 'Failure'
python3 -c 'import torch; print(torch.cuda.is_available())'

time python3 main.py

deactivate
rm -rf rocm-pytorch-2gpus
```

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.


AMD