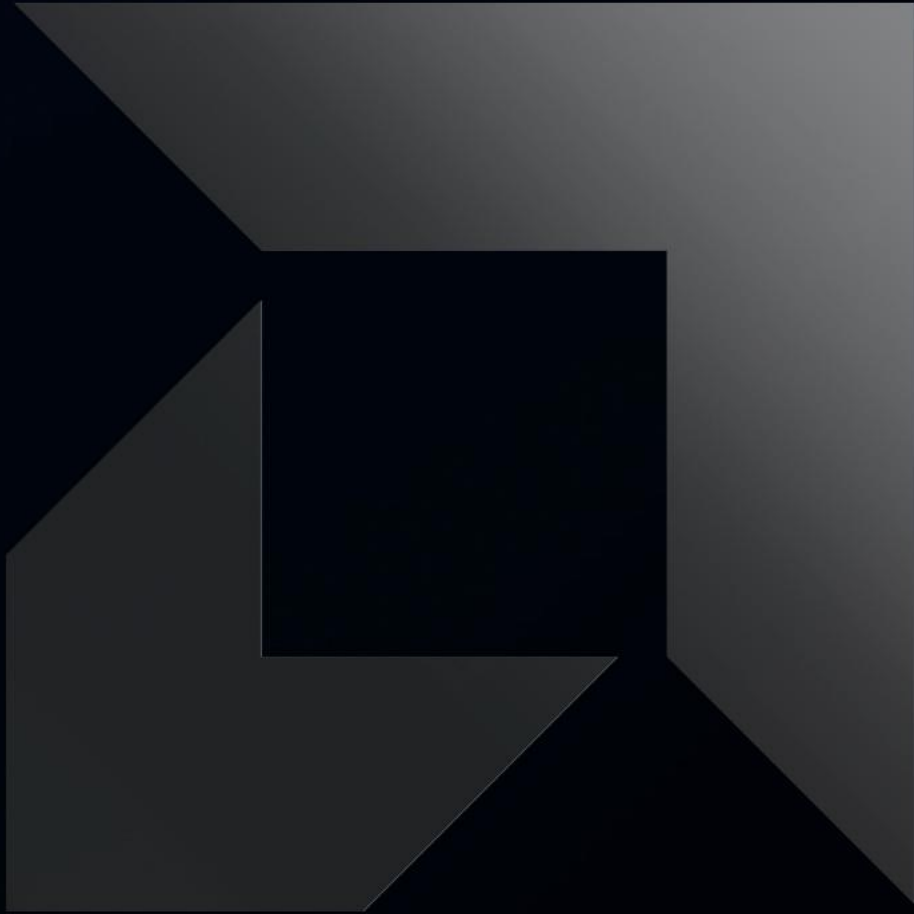




ML and AI on AMD GPUs

Presenter: Bob Robey
AMD @ Tsukuba University
Oct 21-23, 2025

AMD 
together we advance_

ML ECOSYSTEM FOR AMD GPUS

SOURCE & BINARY implementations available upstream!

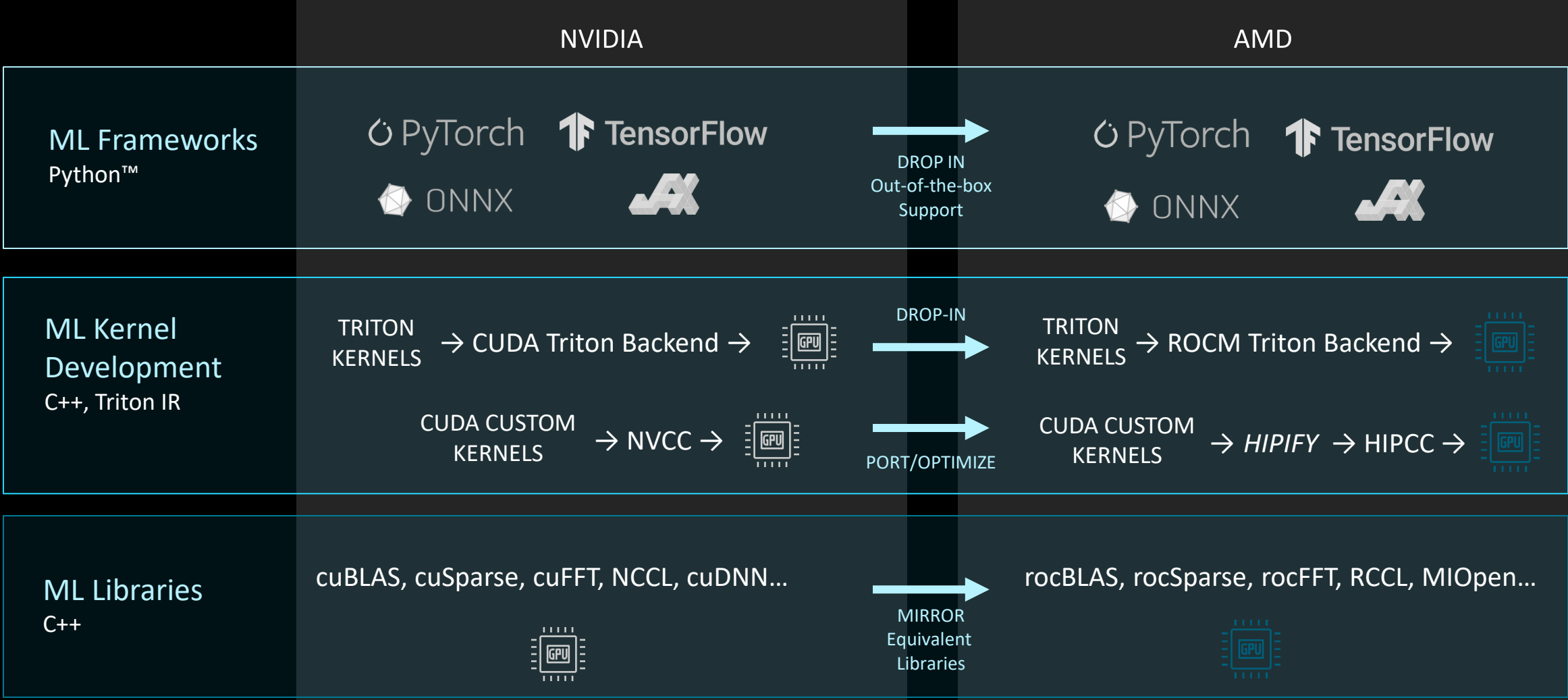


MIOPEN | RCCL | MIVisionX | MIGraphX
DEEPSPEED | AITemplate | openAI Triton | cuPy | XGBoost | hip-Python
flash-attention | vLLM | bitsandbytes



PORTING YOUR ML APPLICATION

NO CHANGES REQUIRED IN MOST CASES



PORTING YOUR ML APPLICATION

API COMPATIBLE LIBRARIES

CUDA Library	ROCm Library	Description
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transform Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	LAPACK Library
AmgX	rocALUTION	Algebraic Multi-Grid Accelerated Linear Solvers
Thrust	hipThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

Linear Algebra Libraries supporting both AMD and NVIDIA GPUs

Eigen	C++ template library for linear algebra
MAGMA	Dense linear algebra library on GPU and Multicore Architectures
SuperLU_DIST	Direct solution of large, sparse, nonsymmetric systems of linear equation
HYPRE	Scalable Linear Solvers and Multigrid Methods
ELPA	Highly efficient and highly scalable direct eigensolvers for symmetric (hermitian) matrices.
...	

WHERE CAN I GET AI FRAMEWORKS FROM?

BINARY AND SOURCE DISTRIBUTIONS AVAILABLE

	Source	Container	PIP Wheel
 TensorFlow	TensorFlow GitHub	Docker Hub	pypi.org
 PyTorch	PyTorch GitHub	Docker Hub	pytorch.org
 ONNX RUNTIME	ONNX-RT GitHub	Docker Instructions	onnxruntime.ai
JAX	GitHub public fork	Docker Hub	ROCm GitHub
DeepSpeed	DeepSpeed GitHub	Docker Hub	deepspeed.ai
CuPy	cupy.dev	Docker Hub	cupy.dev

- ▲ Source builds can sometimes be.....tricky
- ▲ Leveraging containers or provided Wheel piles for Python™ installs is recommended if possible!

WHERE CAN I GET AI FRAMEWORKS FROM?

ROCm GITHUB HOSTS MOST AMD GPU PORTS AND LATEST CHANGES

▲ <https://github.com/ROCm>

**AMD ROCm™ Software**
412 followers <https://rocm.docs.amd.com> Part of AMD

README.md

AMD ROCm™ Software

AMD ROCm software is AMD's Open Source stack for GPU computation.

To learn more about ROCm, check out our [Documentation](#), [Examples](#), and [Developer Hub](#).

If you have questions or need help, reach out to us on GitHub.

Popular repositories

ROCm Public
AMD ROCm™ Software - GitHub Home
Python 4k 327

HIP Public
HIP: C++ Heterogeneous-Compute Interface for Portability
C++ 3.4k 514

MIOpen Public
AMD's Machine Intelligence Library

tensorflow-upstream Public
Forked from [tensorflow/tensorflow](#)
TensorFlow ROCm port

ROCm ML FRAMEWORK CONTAINER IMAGES

ROCm DOCKERHUB PROVIDES SEVERAL IMAGES READY TO USE

- Docker® command:

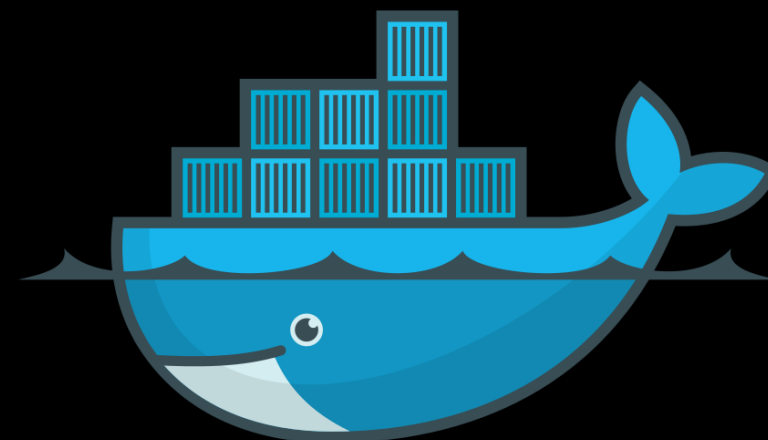
```
sudo docker run -it \  
--network=host \  
--device=/dev/kfd \  
--device=/dev/dri \  
--group-add=video \  
--ipc=host \  
--cap-add=SYS_PTRACE \  
--security-opt seccomp=unconfined \  
rocm/<my favorite ML framework>
```

Allow container to access network (dist. learning)

Allow access to the GPU drivers

Security options - may need
to be expanded depending
on the host system
configuration

Replace by your image of choice



- Some of the available images:

- PyTorch <https://hub.docker.com/r/rocm/pytorch>: *rocm/pytorch:latest*
- TensorFlow <https://hub.docker.com/r/rocm/tensorflow>: *rocm/tensorflow:latest*
- Jax <https://hub.docker.com/r/rocm/jax>: *rocm/jax:latest*

- Docker images with different tags/versions are available on ROCm DockerHub
- Apptainer and Podman can also be used

Prerequisites

- System dependencies need to be met:
 - GPU drivers
 - ROCm libraries, RCCL, and MIOpen should be installed:
 - rocm-dev
 - hiplibsdk
 - mlsdk
- ROCm and Python™ versions must match the wheel file
- <https://rocm.docs.amd.com/en/develop/>

Install

- Linux
 - [Quick start guide](#)
 - [Linux install guide](#)
 - [Package manager integration](#)
- Windows
 - [Windows install guide](#)
 - [Application deployment guidelines](#)
- [Install Docker containers](#)
- [PyTorch for ROCm](#)
- [TensorFlow for ROCm](#)
- [MAGMA for ROCm](#)
- [ROCm & Spack](#)

Example for PyTorch install

- Pytorch official docs: <https://pytorch.org/>

PyTorch Build	Stable (2.2.2)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 5.7	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm5.7</pre>			

Choice for a stable or a preview build
Supporting ROCm version

Suggested install command

- More combinations available! Check here <https://download.pytorch.org/whl/torch/>
 - E.g. PyTorch 2.2.2 with ROCm 5.6
 - Older versions
- More combinations available in <https://repo.radeon.com/rocm/manylinux>.

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 built against.

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

```
pip3 install -t $wd/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
```

Make the freshly installed PyTorch available to your Python runs

```
PYTHONPATH=$wd/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.

PyTorch install – virtual environments

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

Leverage the venv module to create the virtual environment

We are happy to leverage system's already installed packages



```
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())'
```

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.

```
curl -LO https://repo.anaconda.com/miniconda/Miniconda3-py310_24.1.2-0-Linux-x86_64.sh
```

```
bash ./Miniconda3-* -b -p miniconda3 -s
```

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

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

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

Download and install a minimal conda (miniconda).

Activate the conda environment

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())'
```

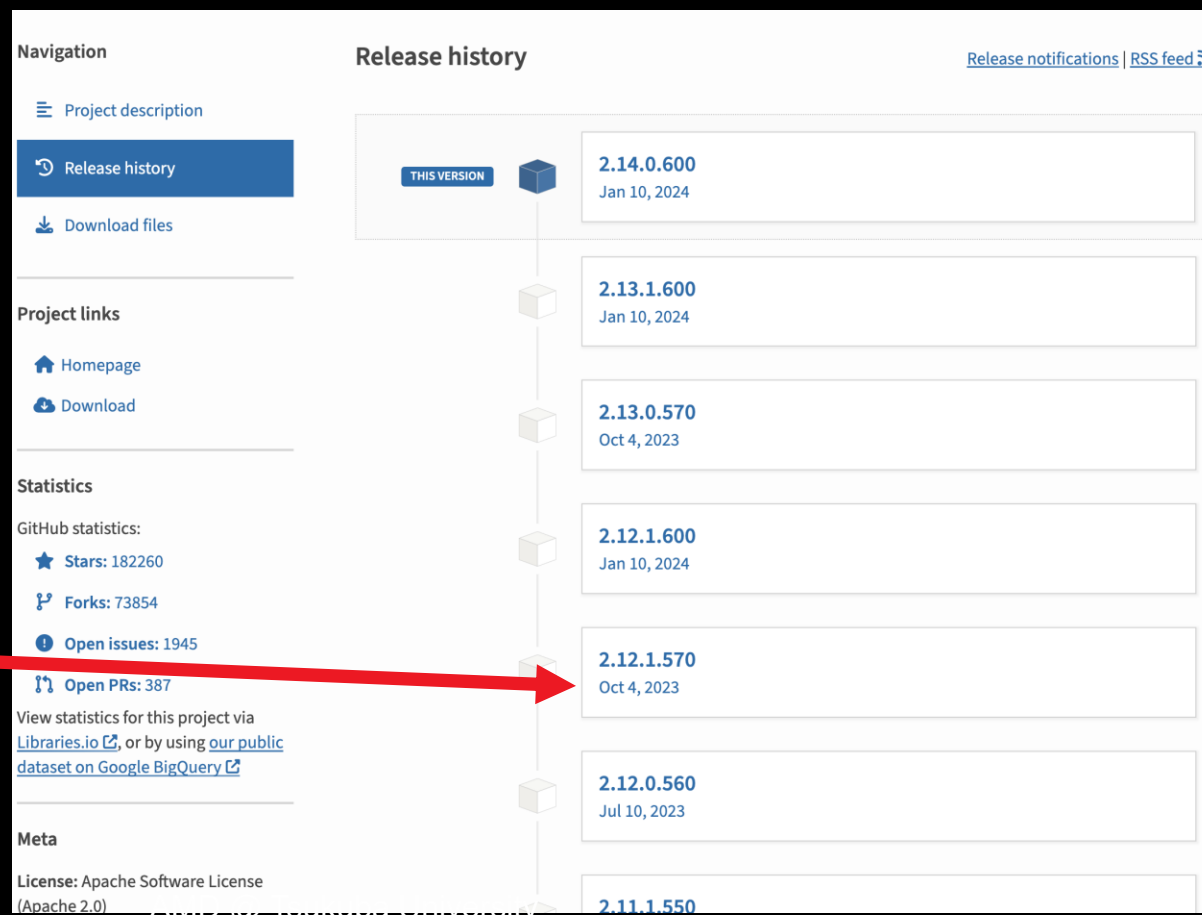
TensorFlow install

- Requirements similar to PyTorch install.
- Unlike PyTorch, TensorFlow packages for ROCm have a specific name:
 - **tensorflow-rocm**

Check the available versions from
<https://pypi.org/project/tensorflow-rocm/#history>

Package version example:

- TensorFlow version 2.12.1
- Built on top of ROCm 5.7.0



TensorFlow install

- Installing TensorFlow 2.12.0 for ROCm 5.6.x:

```
pip install tensorflow-rocm==2.12.0.560
```

- One can leverage pip command to get the available versions

```
pip install tensorflow-rocm==
```

```
ERROR: Could not find a version that satisfies the requirement tensorflow-rocm== (from versions: 2.8.0, 2.8.1, 2.8.2, 2.8.3, 2.8.4, 2.9.1, 2.9.2, 2.9.3, 2.9.4, 2.10.0.520, 2.10.0.530, 2.10.1.540, 2.10.1.550, 2.11.0.530, 2.11.0.540, 2.11.1.550, 2.12.0.560, 2.12.1.570, 2.12.1.600, 2.13.0.570, 2.13.1.600, 2.14.0.600)
```

- Checking number of GPUs:

```
python -c 'from tensorflow.python.client import device_lib ; device_lib.list_local_devices()'
```

Created device /device:GPU:0 with 63922 MB memory: -> device: 0, name: AMD Instinct MI210, pci bus id: 0000:63:00.0

Created device /device:GPU:1 with 63922 MB memory: -> device: 1, name: AMD Instinct MI210, pci bus id: 0000:43:00.0

Created device /device:GPU:2 with 63922 MB memory: -> device: 2, name: AMD Instinct MI210, pci bus id: 0000:03:00.0

Created device /device:GPU:3 with 63922 MB memory: -> device: 3, name: AMD Instinct MI210, pci bus id: 0000:26:00.0

JAX install

- JAX GPU support comes from the package `jaxlib`
- `jaxlib` requires building from source upstream
- You can leverage the wheel files from ROCm Github:
 - <https://github.com/ROCm/jax/releases>

Jax/Jaxlib 0.4.26 release for ROCm Latest

****Jax/Jaxlib 0.4.26 release for ROCm**

Install Jaxlib
Python 3.9

```
python3 -m pip install https://github.com/ROCmSoftwarePlatform/jax/releases/download/jaxlib-v0.4.26/
```

```
python3 -m pip install https://github.com/ROCmSoftwarePlatform/jax/releases/download/jaxlib-v0.4.26/
```

Python 3.10

```
python3 -m pip install https://github.com/ROCmSoftwarePlatform/jax/releases/download/jaxlib-v0.4.26/
```

```
python3 -m pip install https://github.com/ROCmSoftwarePlatform/jax/releases/download/jaxlib-v0.4.26/
```

JAX install

- Example installing commands:

```
pip install \
  https://github.com/ROCmSoftwarePlatform/jax/releases/download/jaxlib-v0.4.26/
  jaxlib-0.4.26+rocm600-cp310-cp310-manylinux2014_x86_64.whl
pip install jax==0.4.26
```

← JAX package doesn't have any GPU software dependency

- Checking number of GPUs:

```
python -c 'import jax ; print("I have this many GPUs:", jax.local_device_count())'
```

- Should yield depending on the system:

I have this many GPUs: 8

PyTorch source install – see HPCTrainingDock script

- Build PyTorch using PyTorch ROCm Docker® image: *rocm/pytorch:latest-base*

```
# Get latest PyTorch docker image - it contains all dependencies needed.
```

```
docker pull rocm/pytorch:latest-base
```

```
# Start the container
```

```
docker run -it \  
--cap-add=SYS_PTRACE --security-opt seccomp=unconfined \  
--device=/dev/kfd --device=/dev/dri --group-add video --ipc=host \  
--shm-size 8G \  
rocm/pytorch:latest-base
```

```
# Clone the desired version of PyTorch
```

```
git clone --recursive https://github.com/pytorch/pytorch.git
```

```
# Build targeting a given GPU architecture.
```

```
cd pytorch ; PYTORCH_ROCM_ARCH=gfx90a ./ci/pytorch/build.sh
```

- Should you prefer not to rely on existing PyTorch ROCm Docker image:

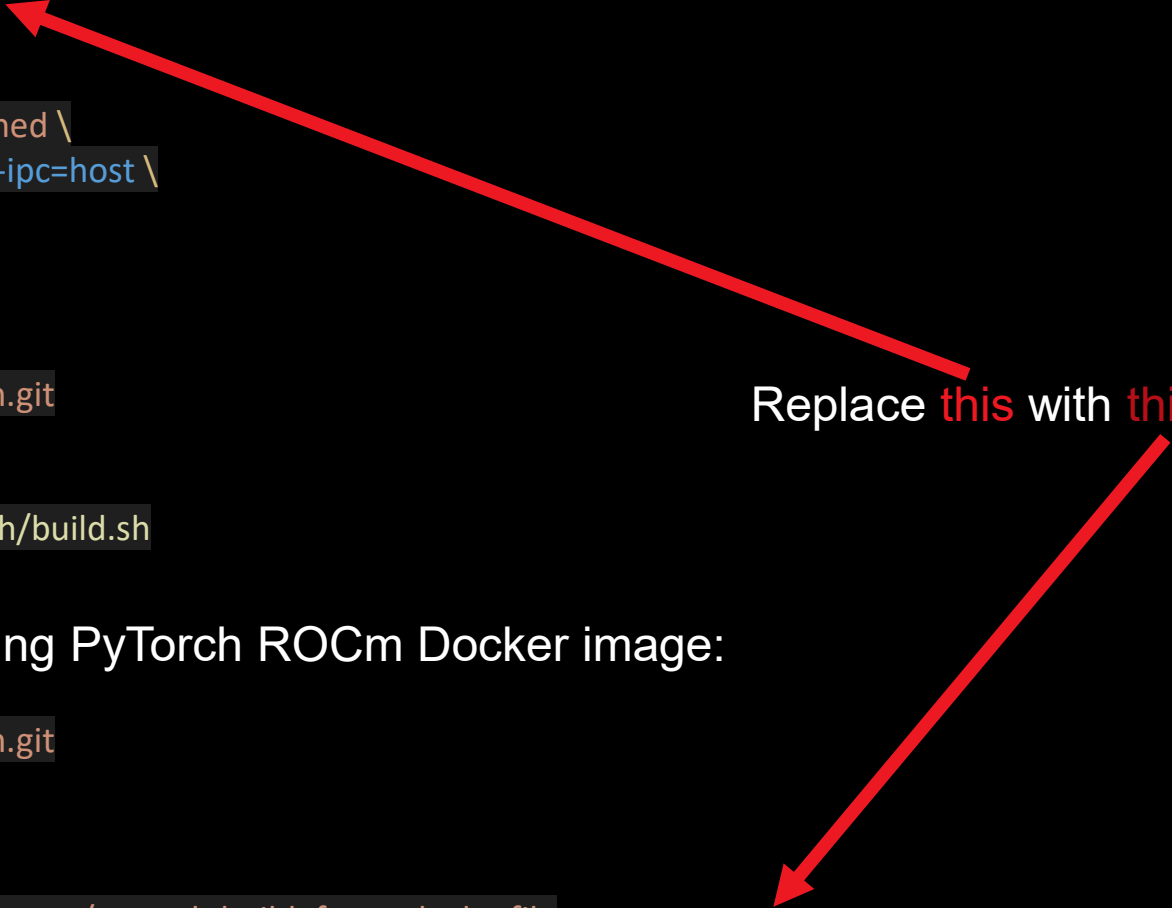
```
# Clone the desired version of PyTorch
```

```
git clone --recursive https://github.com/pytorch/pytorch.git
```

```
# Build a docker image with the desired version.
```

```
cd pytorch/ci/docker  
./build.sh pytorch-linux-ubuntu22.04-rocm6.0-py3.10 -t rocm/pytorch:build_from_dockerfile
```

Replace **this** with **this**



TensorFlow source install

- Build TensorFlow leveraging ROCm Docker® image, e.g. rocm/dev-ubuntu-22.04:6.0-complete

Start the container

```
docker run -it \  
--cap-add=SYS_PTRACE --security-opt seccomp=unconfined \  
--device=/dev/kfd --device=/dev/dri --group-add video --ipc=host \  
--shm-size 8G \  
rocm/dev-ubuntu-22.04:6.0.2-complete
```

Clone desired version of Pytorch

```
git clone --recursive -b r2.14-rocm-enhanced \  
https://github.com/ROCm/tensorflow-upstream
```

Refer to repo available docs: tensorflow-upstream/rocm-doc/tensorflow-build-from-source.md

Install system and Python™ requirements

```
apt-get install python3-dev python3-pip openjdk-8-jdk openjdk-8-jre unzip wget git  
pip3 install numpy wheel mock future pyyaml setuptools requests keras_preprocessing keras_applications  
jupyter
```

Install bazel version from <https://github.com/bazelbuild/bazel/releases>

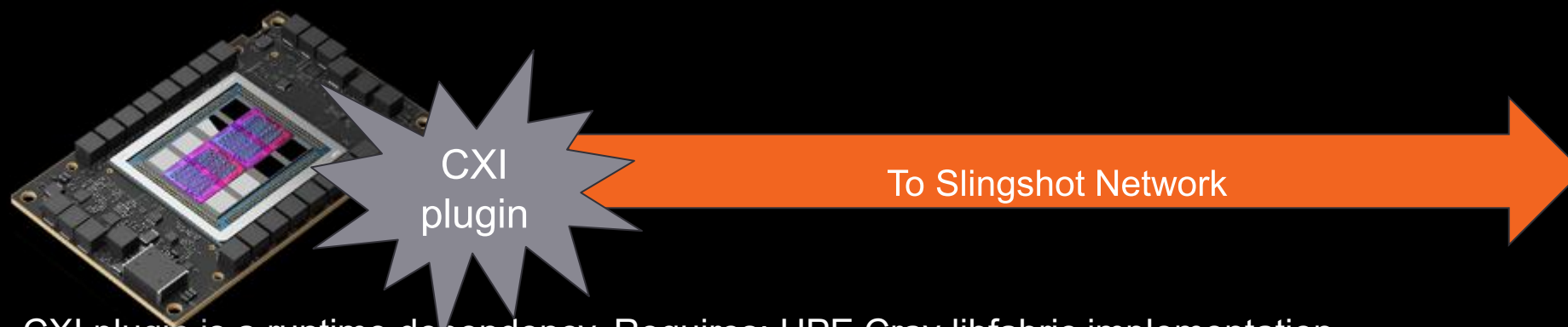
```
cat tensorflow-upstream/.bazelversion
```

Build: result wheel file will be in /tmp/tensorflow_pkg

```
cd tensorflow-upstream \  
./build_rocm_python3.sh
```

Comms are important! - RCCL AWS-CXI plugin

- LUMI, Frontier (and others) directly attaches AMD Instinct™ MI250x Accelerator to the Cray 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

- **CXI plugin is NOT featured in official DockerHub images! Make sure it exists 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
```

Horovod install

- 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
```





deepspeed

DeepSpeed install

- 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"
```

Select all the optimizations
not all are enabled for
GPUs.

Allow multiple process builds.

ds_report

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]
-----
```

PyTorch example

- 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

Frameworks tend to include information on how the training is distributed.

- ... 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

TensorFlow + Horovod example

- What provides distributed capability: Horovod wraps the TensorFlow operators

```
import tensorflow as tf
import horovod.tensorflow as hvd
```

Instantiate Horovod implementation for TensorFlow

```
# Horovod: initialize Horovod.
hvd.init()
```

Horovod requires an initialization similar to what MPI requires: it will leverage MPI dependency to record the information about the ranks

```
# Horovod: pin GPU to be used to process local rank (one GPU per process)
tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

```
# A local optimizer.
opt = tf.optimizers.SGD(0.01)
```

Horovod provides information about local rank useful for GPU binding.

```
# Wrap the Tensorflow operator (e.g. a gradient tape) with the corresponding
# Horovod distributed operator.
tape = tf.GradientTape()
tape = hvd.DistributedGradientTape(tape, compression=compression)
```

```
# Apply the gradient information gather from the distributed operator into the local optimizer.
gradients = tape.gradient(loss, model.trainable_variables)
opt.apply_gradients(zip(gradients, model.trainable_variables))
```

Leverage the results of the distributed operator in the local optimizer.

DISCLAIMERS 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's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

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.

© 2025 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon™, Instinct™, EPYC, Infinity Fabric, ROCm™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

Apache and the Apache feather logo are trademarks of The Apache Software Foundation.

Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc.

HPE is a registered trademark of Hewlett Packard Enterprise Company and/or its affiliates.

