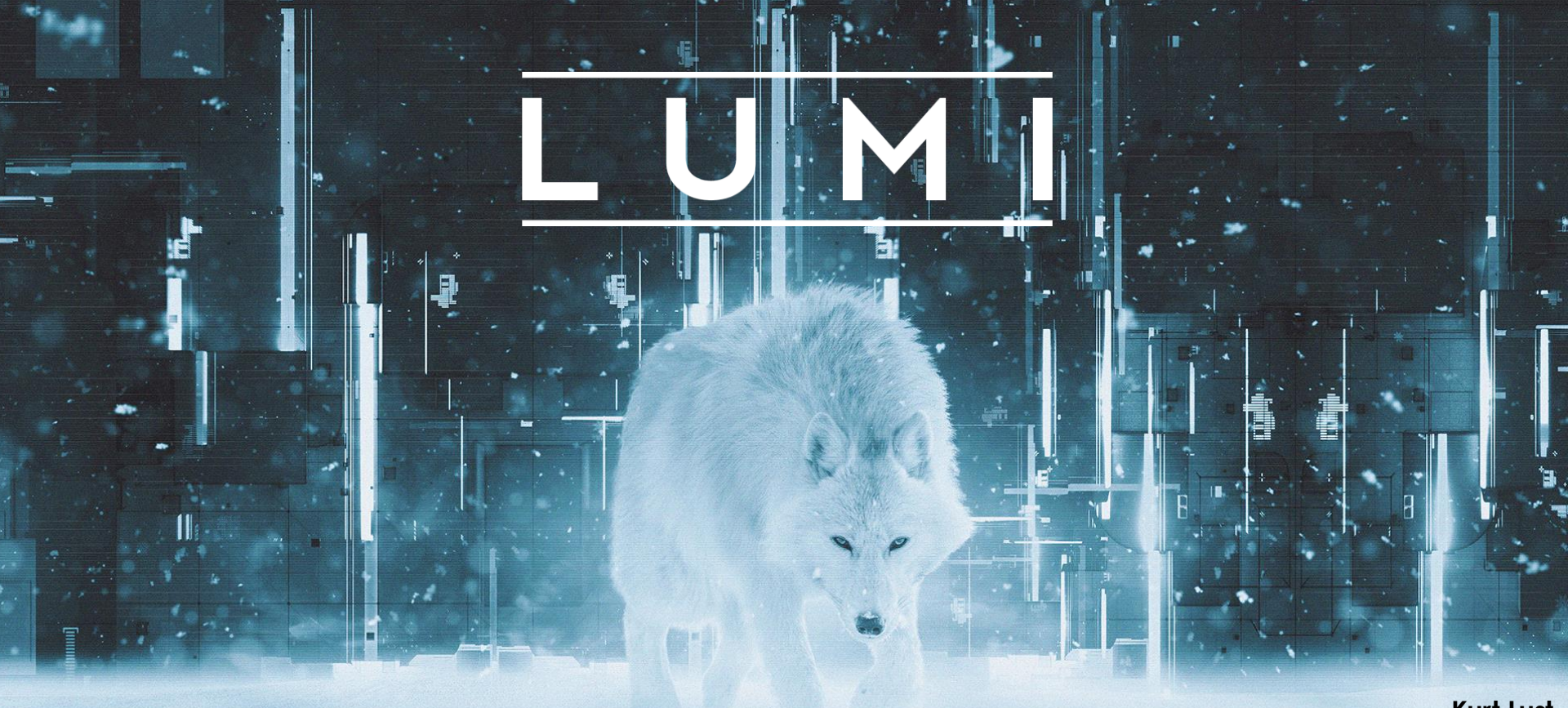

Former AI containers on LUMI-G

Kurt Lust
LUMI User Support Team (LUST)
University of Antwerp

Evolving version, last update March 2026

Former prebuilt containers for AI (and some others)

- AI containers used to be built by AMD for LUMI until the January 2026 maintenance:
  - PyTorch: Best tested
  - JAX
  - TensorFlow
  - AlphaFold
  - ROCm and mpi4py
- Where to find?
  - `/appl/local/containers/sif-images`: Links to the latest version of each container
  - `/appl/local/containers/easybuild-sif-images`: Images for EasyBuild
    - Recommended for inexperienced users, but work-in-progress
  - `/appl/local/containers/tested-containers`: Images linked to and docker tarballs
- Recommend to keep your own copy of the image you depend upon!

Running the AI containers (Complicated way)

- The containers have everything they need to use RCCL and/or MPI on LUMI
- Need to take care of bindings:
  - Need `-B /var/spool/slurmd,/opt/cray`, and older ones `/usr/lib64/libcxi.so.1` at the minimum (and this list may change after a system update or changes in the container builds)
  - And add access to your space in `/project`, `/scratch` and/or `/flash` (default is only the home directory):
    `-B /pfs,/scratch,/projappl,/project,/flash`
- Components that need further initialisation:
  - MIOpen (the AMD cuDNN equivalent)
  - RCCL needs to be told the right network interfaces to use if you run across nodes
  - GPU-aware MPI may need to be set up (see earlier in the course)
  - Your AI package may need some too (e.g., `MASTER_ADDR` and `MASTER_PORT` for distributed learning with PyTorch)
- Containers with Python packages are built using Conda
  - Need to initialise the Conda environment via `$WITH_CONDA` in older versions of the container

Running the AI containers
EasyBuild (1)

- We provide EasyBuild recipes to "install" some of the containers and provide a module.
  - For those packages for which we know generic usage patterns, we provide some scripts that do most settings, and new PyTorch containers have scripts equivalent to the CSC ones
  - Define a number of environment variables to make life easier, e.g., popular bindings and a variable referring to the container
  - All but the very oldest versions come with a Python virtual environment pre-initialised to add your own packages
    - No more `$WITH_CONDA` needed as the module takes care of injecting environment variables in the container that have the same effect as the Conda and Python virtual environment activate scripts
    - Management of the Python virtual environment: Create a SquashFS file from the installation
- Someone with some EasyBuild experience may further extend the recipe to, e.g., already install extra packages

Running the AI containers EasyBuild (2)

- Install:
  - Set up your user environment for EasyBuild (`EBU_USER_PREFIX`)
  - Run
    ```
    module load LUMI partition/container EasyBuild-user
    eb PyTorch-2.6.0-rocm-6.2.4-python-3.12-singularity-20250404.eb
    ```
  - After that the container module is available in all LUMI stacks and in CrayEnv
- Best to clean up afterwards before running (or take a new shell)
- Will copy the .sif-file to the software installation directory.
  - To delete:
    ```
    module load PyTorch/2.6.0-rocm-6.2.4-python-3.12-singularity-20250404
    rm –f $SIF
    module load PyTorch/2.6.0-rocm-6.2.4-python-3.12-singularity-20250404
    ```
  - At your own risk as we may remove the image in `/appl/local/containers` without notice

Running: Example: Distributed learning Without EasyBuild (1)

- Create file `get-master.py`:

```
import argparse
def get_parser():
    parser = argparse.ArgumentParser(description="Extract master node name from Slurm node list",
             formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument("nodelist", help="Slurm nodelist")
    return parser

if __name__ == '__main__’:
    parser = get_parser()
    args = parser.parse_args()

    first_nodelist = args.nodelist.split(',')[0]

    if '[' in first_nodelist:
        a = first_nodelist.split('[‘)
        first_node = a[0] + a[1].split('-')[0]

    else:
        first_node = first_nodelist

    print(first_node)
```

Running: Example: Distributed learning Without EasyBuild (2)

- Create file `run-pytorch.sh`:

```
#!/bin/bash -e

# Make sure GPUs are up
if [ $SLURM_LOCALID -eq 0 ] ; then
    rocm-smi
fi
sleep 2

$WITH_CONDA

# Set MIOpen cache to a temporary folder.
export MIOPEN_USER_DB_PATH="/tmp/$(whoami)-miopen-cache-$SLURM_NODEID"
export MIOPEN_CUSTOM_CACHE_DIR=$MIOPEN_USER_DB_PATH

if [ $SLURM_LOCALID -eq 0 ] ; then
    rm -rf $MIOPEN_USER_DB_PATH
    mkdir -p $MIOPEN_USER_DB_PATH
fi
sleep 2

# Set ROCR_VISIBLE_DEVICES so that each task uses the proper GPU
export ROCR_VISIBLE_DEVICES=$SLURM_LOCALID

# Report affinity
echo "Rank $SLURM_PROCID --> $(taskset -p $$)"

# Set interfaces to be used by RCCL.
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3

# Set environment for the app
export MASTER_ADDR=$(python get-master.py "$SLURM_NODELIST")
export MASTER_PORT=29500
export WORLD_SIZE=$SLURM_NPROCS
export RANK=$SLURM_PROCID

# Run app
python -u mnist_DDP.py --gpu --modelpath model
```

Check for the GPUs

Initialise Conda

MIOpen configuration

GPU binding

RCCL configuration

Who's the master?

Ready to run…

Running: Example: Distributed learning Without EasyBuild (3)

- Create job script `my-job.sh`:

```
#!/bin/bash -e
#SBATCH --nodes=4
#SBATCH --gpus-per-node=8
#SBATCH --tasks-per-node=8
#SBATCH --output="output_%x_%j.txt”
#SBATCH --partition=standard-g
#SBATCH --mem=480G
#SBATCH --time=00:10:00
#SBATCH --account=project_<your_project_id>

PROJECT_DIR=/project/your_project/your_directory
SIF=/appl/local/containers/easybuild-sif-images/lumi-pytorch-rocm-6.2.4-python-3.12-pytorch-v2.6.0-dockerhash-36e16fb5b67b.sif

c=fe
MYMASKS="0x${c}000000000000,0x${c}00000000000000,0x${c}0000,0x${c}000000,0x${c},0x${c}00,0x${c}00000000,0x${c}0000000000”

srun --cpu-bind=mask_cpu:$MYMASKS \
  singularity exec \
    -B /var/spool/slurmd \
    -B /opt/cray \
    -B /usr/lib64/libcxi.so.1 \
    -B $PROJECT_DIR:/workdir \
    $SIF /workdir/run-pytorch.sh
```

Reorder CPU slots for easy GPU binding

Run the script from the previous slide

Running: Example: Distributed learning With EasyBuild-installed module

- Create job script `my-job.sh`:

```
#!/bin/bash -e
#SBATCH --nodes=4
#SBATCH --gpus-per-node=8
#SBATCH --tasks-per-node=8
#SBATCH --output="output_%x_%j.txt”
#SBATCH --partition=standard-g
#SBATCH --mem=480G
#SBATCH --time=00:10:00
#SBATCH --account=project_<your_project_id>

module load CrayEnv PyTorch/2.3.1-rocm-6.0.3-python-3.12-singularity-20240923

c=fe
MYMASKS="0x${c}000000000000,0x${c}00000000000000,0x${c}0000,0x${c}000000,0x${c},
0x${c}00,0x${c}00000000,0x${c}0000000000”

srun --cpu-bind=mask_cpu:$MYMASKS \
  singularity exec $SIF \
    conda-python-distributed -u mnist_DDP.py --gpu --modelpath model
```

Extending container 1: cotainr

- It is possible to use the ROCm containers in `/appl/local/containers/sif-images` as a base image for cotainr and build your own AI container
  - Be careful which version of the AI software you use as wheels are likely for a specific ROCm version (and you don't want to pick up wheels for NVIDIA)
  - MPI may be a problem as mpi4py has to come from Conda
- Process:
  - Create a yaml file with the setup for Conda (see notes)
  - Run cotainr:

```
module load LUMI/24.03 cotainr
cotainr build my-new-image.sif \
    --base-image=/appl/local/containers/sif-images/lumi-rocm-rocm-6.0.3.sif \
    --conda-env=py312_rocm603_pytorch.yml
```

- Run as a regular container
  - Or find someone who want to make an EasyConfig to create a module and point EasyBuild to the container .sif file with `--sourcepath`

Extending container 2: singularity build

- Build a singularity-compatible container definition file, e.g.,

```
Bootstrap: localimage

From: /appl/local/containers/easybuild-sif-images/lumi-pytorch-
rocm-6.0.3-python-3.12-pytorch-v2.3.1-dockerhash-2c1c14cafd28.sif

%post

zypper -n install -y Mesa libglvnd libgthread-2_0-0 hostname
```

- And run:

```
module load LUMI/24.03 systools
singularity build my-new-container.sif my-container-definition.def
```

- Good way to add SUSE packages that may be needed to install extra software
- Tip: Start from a container with an EasyBuild module and the module might still work…

Extending container 3: Python virtual environment (1)

- All but the oldest containers installed with EasyBuild have a pre-initialised virtual environment
  - In the container available as `/user-software/venv/<MyVEnv>`
  - Outside the container: `$CONTAINERROOT/user-software/venv/<MyVEnv>`
  - And `/user-software` can also be used to install other software if needed…
- How?

```
$> module load LUMI
$> module load PyTorch/2.6.0-rocm-6.2.4-python-3.12-singularity-20250404
$> singularity shell $SIF
Singularity> pip install pytorch-lightning
```

Extending container 3: Python virtual environment (2)

- But what about the many small files?
  - Convert `$CONTAINERROOT/user-software` to a SquashFS file
    `make-squashfs`
    And reload the module…
  - You can then delete the `$CONTAINERROOT/user-software` subdirectory if you need the space (or file quota) and reconstruct it if needed with `unmake-squashfs`
  - To add additional packages afterwards:
    - Make sure the `$CONTAINERROOT/user-software` exists (outside the container)
    - Delete `$CONTAINERROOT/user-software.squashfs`
    - Reload the module
    - And start a shell in the container…
- You can of course do this with any container with Python, also when not using EasyBuild-built modules but the manual procedure takes a few more steps.


Questions?