Usage instructions for AMD MI300X nodes (gaod0001 and gaod0002)

Version: 0.2
Gabriele Inghirami
13 May 2025

Contents

1 Introduction

This document serves as a basic user guide for the compute nodes equipped with AMD MI300X GPUs at the Lichtenberg II high-performance cluster at TU-Darmstadt. In the cluster there are two such nodes available, named gaod0001 and gaod0002, each with 8 accelerators (i.e., GPUs). For the technical point of view, they are Lenovo Thinkserver SR685a V3.The OS of these nodes is Red Hat Enterprise Linux version 9.4. You can check the release with "`cat /etc/redhat-release`".
At the time of writing, there are frequent updates and changes in many different contexts: in the PyTorch and Tensorflow frameworks, in the AMD support and in how we manage the software stack. Therefore, **we recommend to always check and use the latest version of this document**.

2 Allocation of computing time with Slurm

As in the case of the other computing nodes, to use the servers you first need to reserve computing time via the Slurm workload manager. If you are not familiar with Slurm on the Lichtenberg Cluster, please, read this page first and have a look at the slides used during the

introduction training events.
The AMD MI300X nodes are grouped into the *special_mi300_x* partition. The parameter to select how many GPUs to request via the "Generic RESource scheduling" is *gpu:mi300x:N*, where N is the number of requested GPUs within a node and it varies from 1 to 8 (even if you want to use multiple nodes).
The nodes are assigned by Slurm "not exclusively", i.e. multiple users can access the resources of the node at the same time, as long as there are some left. For example, a node could be used simultaneously by 3 users, one requesting 4 GPUs and the other two 2 GPUs each. So, please, always pay attention to request only the resources that you will really use.

2.1 Non-interactive (batch) usage

A Slurm batch script to request 4 MI300X GPUs should contain the lines:

```
#SBATCH --partition=special_mi300x
#SBATCH --gres=gpu:mi300x:4
```

If you need a different number of GPUs, let's say N, change gpu:mi300x:4 to gpu:mi300x:N.

2.2 Interactive usage

To interactively use an AMD server, for example to build/install additional software, you can use something like:

```
srun -t 02:00:00 -n1 -c8 --mem-per-cpu=8G -p special_mi300x --gres=gpu:mi300x:4 --pty /bin/bash
```

This command requests two hours (*-t 02:00:00*) of computing time to execute 1 task (*-n1*), with 8 CPU cores (*-c8*), with 8 GBytes of RAM per CPU (so, in this case, 64 GB in total) (*–mem-per-cpu=8G*), on the cluster partition dedicated to the AMD MI 300X nodes (*-p special_mi300x*), using 4 GPUs (*–gres=gpu:mi300x:4*), to run the bash shell interactively (*–pty /bin/bash*).

3 ROCm

AMD provides a rich software stack, called ROCm, *"composed primarily of open-source software, that provides the tools for programming AMD Graphics Processing Units (GPUs), from low-level kernels to high-level end-user applications."*
Based on our experience in the first months of usage and on the recommendations by the Lenovo Support, we strongly suggest to use the ROCm suite that we have installed in the /opt/rocm directory, although it might not be the latest version available, because it is very important that the features offered by the ROCm are adequately supported by the underlying firmware and by the hardware drivers.
At the time of writing, the AMD nodes use the ROCm version 6.4.0.

3.1 Getting information about the GPUs

To quickly check if the GPUs are currently used by the system, you can use the simple command:

```
rocm-smi
```

which should return something like:

```
================================================== ROCm System Management Interface ==================================================
============================================================ Concise Info ============================================================
Device  Node  IDs              Temp        Power     Partitions          SCLK    MCLK    Fan  Perf              PwrCap  VRAM%  GPU%
              (DID,     GUID)  (Junction)  (Socket)  (Mem, Compute, ID)
======================================================================================================================================
0       6     0x74a1,   28851  47.0°C      132.0W    NPS1, SPX, 0        126Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
1       3     0x74a1,   43178  41.0°C      123.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
2       2     0x74a1,   32898  47.0°C      125.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
3       5     0x74a1,   22683  43.0°C      124.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
4       8     0x74a1,   53458  44.0°C      130.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
5       9     0x74a1,   2251   41.0°C      126.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
6       7     0x74a1,   8419   48.0°C      135.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
7       4     0x74a1,   63738  43.0°C      125.0W    NPS1, SPX, 0        120Mhz  900Mhz  0%   perf_determinism  750.0W  0%     0%
======================================================================================================================================
======================================================== End of ROCm SMI Log =========================================================
```

If no devices are shown, then the AMD compute node has indeed a problem and you should inform us via the ticketing system.
The GPUs that are currently executing code instructions should display a non-zero value in the last two columns (and normally higher values in other columns, like Power, Temp, SCLK, MCLK and Fan).
The **rocm-smi** command above shows all the GPUs available in the node, not only the GPUs that are at your disposal via the Slurm allocation.
The **amd-smi** command provides even more information. Please, refer to this official overview to know more about it.

3.2 A quick note about ROCm Docker containers

AMD provides several Docker container images on https://www.amd.com/en/developer/resources/infinity-hub.html (preliminary registration needed) and https://hub.docker.com/u/rocm.
Unfortunately, in most cases root privileges are required to properly run them, therefore normally it is not possible to launch them with **podman**, the unprivileged alternative to Docker available for non-administrative users of the cluster.

4 Modules

Just like the other compute nodes, most of the software is accessible by loading/unloading *modules*.
Here you can find a short introduction about the module system on the Lichtenberg II cluster.
At the time of writing, we are working on the upgrade of the operative system of cluster and not all the software which is available in the module system for Red Hat 8.10 is already available in the module system for Red Hat 9.4.
Setting the environment variable MODULEPATH to */shared/module* allows to continue to use the "old" module system also on nodes running Red Hat 9.4, albeit we do not offer any guarantee that the loaded module will work. In many cases the different library versions might lead to a program crash, nevertheless this approach might work for some programs, especially in the case of commercial software.

```
export MODULEPATH=/shared/modules
```

The default directory of the modules for Red Hat 9.4 is */shared/spmodule*, while */shared/spmodule_2025_03* contains the modules which are currently under construction and should not be used for production work, but only for testing at your own risk (the modules there might be completely broken, have serious shortcomings or they could be replaced at any time without notice).

5 Spack

Spack is a package management tool that helps in providing a wide variety of software with different versions for multiple environments.
You can decide to use the Spack installation already available in the system or resort to a completely independent personal setup.
To create your own environment with additional software starting from the system installation, please, follow this short introduction.
For more detailed information about Spack, we refer to the official documentation and tutorial.

To get a list of the available software, type: “spack find -l”.
To get a list of the available versions of a specific software, for example *intel-oneapi-mkl*, type: “spack find -L intel-oneapi-mkl”.
To load a package, type: “spack load <name-of-the-package>”. When multiple versions of the same package are available, it is necessary to specify the desired one and possibly also the compiler which was use to build the software, for example: “spack load openmpi@5.0.3%gcc@11.4.1” (please, note that this is just an example and this specific package might not exist when you try the command). However, sometimes there might still be a residual ambiguity, due to different options or dependencies which were activated at compile time, that can solved by indicating the hash of the package: “spack load \abcdef” (assuming that *abcdef* is the hash of the package).
We are currently preparing the software stack of the cluster for Red Hat 9.4 using Spack, with the aim to make most of the Spack packages also available as modules.

6 Python

By default, Red Hat Linux 9.4 offers Python 3.9.18.
If you need a more recent version (currently 3.11.9 and 3.12.5 are available), you can use either *module* or *spack* to load it. We suggest to install also the python-venv module, needed to use virtual environments, which is the recommended approach if you plan to install additional software with pip.[1]

6.0.1 Load Python 3.11

To load Python 3.11.9 via the module system, we just type:

```
module load python/3.11.9-ixrm
module load python-venv/1.0-gbl2
```

This is the default Python version in the module system, needed as dependency by almost all the Python package modules.

[1] If you are not familiar with pip, you can easily find plenty of information on the internet using a search engine. Asking (with care!) a LLM might also help.

6.0.2 Load Python 3.12

To load Python 3.12 via the module system, we just type:

```
module load python/3.12.5-ofxr
module load python-venv/1.0-cwqf
```

6.1 Python virtual environments

To keep your working environment clean and to easily manage the installation of different Python packages with conflicting dependencies or different versions of the same Python software, it is strongly recommended to use virtual environments. Here below you can find the syntax to create an environment called "your_environment".

```
python -m venv <your_environment>
source <your_environment>/bin/activate
```

The environment resides in a directory having the same name and containing all its files, including the python packages installed with pip when it is active. Please, note that you need to create the environment only once (first line of Code 6.1), but you need to activate it every time you start a pseudo-terminal before you use it (second line in the code above).
To deactivate a python virtual environment, simply type "deactivate".
In the next subsections, we will create two virtual environments, one for PyTorch and the other one for Tensorflow.
In general, it is a good idea to keep updated **pip** inside an environment:

```
pip install --upgrade pip
```

6.2 Temporary directories

When installing a new package, both Spack and pip need to store an amount of temporary data that often exceeds the space available in the default temporary directory. Therefore, it is very important to define the variable **TMPDIR**, expressing the path of a directory mounted on a device with sufficient space. If enough memory has been allocated, using a subdirectory in */dev/shm*, which is actually a RAM disk, generally offers the best performance. However, if the space is not enough (if you don't know it in advance, when it ends you will get the error "No space left on device"), then you should create a temporary subdirectory in your personal *scratch* directory under */work/scratch*, which, by default, has a total size of 20 TB.
For example:

```
export TMPDIR=/dev/shm/tmp_${USER}
mkdir -p ${TMPDIR}
```

or

```
export TMPDIR=/work/scratch/${USER}/tmp_directory
mkdir -p ${TMPDIR}
```

6.3 PyTorch

For PyTorch, we follow the procedure described in this page of the AMD website.

With the commands:

```
export TMPDIR=/dev/shm/tmp_${USER}; mkdir -p ${TMPDIR}
module load python/3.11.9-ixrm
module load python-venv/1.0-gbl2
python -m venv pytorch_env
source pytorch_env/bin/activate
pip install --upgrade pip
pip3 install --pre torch torchvision torchaudio \
--index-url https://download.pytorch.org/whl/nightly/rocm6.4/
```

we:

1. set up a temporary directory
2. load Python 3.11 and its virtual environment module

3. create a python virtual environment named *pytorch_env*
4. activate the python virtual environment that we just created
5. update pip
6. install the latest version of Py-Torch (and its most common packages) built for the ROCm 6.4

For a quick check that the installation went fine, you can launch python and execute:

```
import torch
print(torch.cuda.is_available())
num_devices = torch.cuda.device_count()
print(f"Number of CUDA devices: {num_devices}")
print(f'device name [0]:', torch.cuda.get_device_name(0))
```

You should get something like:

```
[gi67ceri@gaod0002 ~]$ source pytorch-env/bin/activate
(pytorch-env) [gi67ceri@gaod0002 ~]$ python
Python 3.9.18 (main, Dec  4 2024, 00:00:00)
[GCC 11.4.1 20231218 (Red Hat 11.4.1-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.cuda.is_available())
True
>>> num_devices = torch.cuda.device_count()
>>> print(f"Number of CUDA devices: {num_devices}")
Number of CUDA devices: 4
>>> print(f'device name [0]:', torch.cuda.get_device_name(0))
device name [0]: AMD Instinct MI300X
```

The number of CUDA devices returned by the script should match the number of GPUs that you have requested via Slurm.

6.4 TensorFlow

To install Tensorflow, we follow the instructions on the AMD website for the ROCm 6.4.0.
The pip wheels Tensorflow package versions provided by AMD for the ROCm 6.4.0 (2.18.1, 2.17.1, 2.16.2 as of 8. May 2025) need Python 3.12.5.
So, after having

- set up a temporary directory
- loaded Python 3.12

we create a Python virtual environment, we activate it and we install Tensorflow 2.18.1 inside it with:

```
export TMPDIR=/dev/shm/tmp_${USER}; mkdir -p ${TMPDIR}
module load python/3.12.5-ofxr
module load python-venv/1.0-cwqf
python -m venv TF_env
source TF_env/bin/activate
pip install --upgrade pip
pip install tensorflow-rocm==2.18.1 \
-f https://repo.radeon.com/rocm/manylinux/rocm-rel-6.4 --upgrade
```

In order to briefly check that the installation went fine, you can launch python and execute: you can execute the short sample python script in the Tensorflow homepage and near the end of the AMD web page:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```