Usage instructions for Intel Ponte Vecchio nodes

TECHNISCHE UNIVERSITÄT DARMSTADT

HRZ Hochschulrechenzentrum

Version: 0.3.0
Gunnar Eifert and Gabriele Inghirami
May 9, 2025

Contents

1 Introduction

This document serves as a basic installation and usage guide for the Intel Data Center GPU Max compute nodes at the Lichtenberg II high-performance cluster at TU-Darmstadt, formerly called also "Ponte Vecchio" nodes and sometimes abbreviated as PVC. There are five nodes available with four accelerators (i.e., GPUs) each, albeit, at the time of writing, only two are available (gpqd0004 and gpqd0005). This guide applies to Red Hat Enterprise Linux version 9.4[1]. 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 Intel GPU support and in how we manage the software stack. Therefore, **we recommend to always check and use the latest version of this document**.
In these notes we will use:

1. the **module** command to load the common basic Intel software
2. the **spack load** command to load a suitable Python installation
3. the **pip** command, in a python environment, to install PyTorch, Tensoflow and some ad-hoc Intel software

[1]Actually, RHEL 9.4 is not officially supported by Intel, but for most applications the GPUs should still work fine.

Important remark: in the rest of this document we assume that you are interactively logged on one of the five Ponte Vecchio nodes (gpqd0001-gpqd0005) and you are using the BASH shell.

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 nodes are grouped into the *special_tp1* partition. The parameter to select how many GPUs to request via the "Generic RESource scheduling" is *gpu:pvc128g:N*, where N is the number of requested GPUs within a node and it varies from 1 to 4 (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 2 users, each requesting 2 GPUs.

2.1 Non-interactive (batch) usage

A Slurm batch script to request 2 PVC GPUs should contain the lines:

Code 1: Extract of a Slurm batch script

```
#SBATCH --partition=special_tp1
#SBATCH --gres=gpu:pvc128g:2
```

If you need a different number of GPUs, let's say N, change gpu:pvc128g:2 to gpu:pvc128g:N.

2.2 Interactive usage

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

Code 2: Time allocation for interactive usage

```
srun -t 02:00:00 -n1 -c4 --mem-per-cpu=8G -p special_tp1 --gres=gpu:pvc128g:2 --pty /bin/bash
```

This command requests two hours (*-t 02:00:00*) of computing time to execute 1 task (*-n1*), with 4 CPU cores (*-c4*), with 8 GBytes of RAM per CPU (so, in this case, 32 GB in total) (*–mem-per-cpu=8G*), on the cluster partition dedicated to the PVC nodes (*-p special_tp1*), using 2 GPUs (*–gres=gpu:pvc128g:2*), to run the bash shell interactively (*–pty /bin/bash*).

2.2.1 Preliminary check

Before to begin using the server interactively, we recommend to preliminary check if all the GPUs are recognized by the operating system.
You can accomplish this task with:

Code 3: Preliminary check

```
 xpu-smi discovery
```

which should return:

```
+-----------+--------------------------------------------------------------------------------------+
| Device ID | Device Information                                                                   |
+-----------+--------------------------------------------------------------------------------------+
| 0         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-003a-0000-002f0bd58086                                       |
|           | PCI BDF Address: 0000:3a:00.0                                                        |
|           | DRM Device: /dev/dri/card0                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 1         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-004b-0000-002f0bd58086                                       |
|           | PCI BDF Address: 0000:4b:00.0                                                        |
```

```
|           | DRM Device: /dev/dri/card1                                                                  |
|           | Function Type: physical                                                                     |
+-----------+---------------------------------------------------------------------------------------------+
| 2         | Device Name: Intel(R) Data Center GPU Max 1550                                              |
|           | Vendor Name: Intel(R) Corporation                                                           |
|           | SOC UUID: 00000000-0000-00ca-0000-002f0bd58086                                              |
|           | PCI BDF Address: 0000:ca:00.0                                                               |
|           | DRM Device: /dev/dri/card2                                                                  |
|           | Function Type: physical                                                                     |
+-----------+---------------------------------------------------------------------------------------------+
| 3         | Device Name: Intel(R) Data Center GPU Max 1550                                              |
|           | Vendor Name: Intel(R) Corporation                                                           |
|           | SOC UUID: 00000000-0000-00da-0000-002f0bd58086                                              |
|           | PCI BDF Address: 0000:da:00.0                                                               |
|           | DRM Device: /dev/dri/card3                                                                  |
|           | Function Type: physical                                                                     |
+-----------+---------------------------------------------------------------------------------------------+
```

If no devices are shown, then the compute node has indeed a problem and you should inform us via the ticketing system, albeit in most cases we should be already aware of the problem and working to fix it.

3 Intel modules

Intel provides a software suite, called Intel One API, whose later version currently installed in the Lichtenberg II cluster can be loaded with:

Code 4: Load required oneAPI packages

```
export MODULEPATH=/shared/modules
module load intel/2025.0
module load intelmpi
```

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, but this approach is meant to be only temporary. When the transition to Red Hat 9.4 will be complete, we hope to adopt a simpler and more uniform approach.

To better understand the software's possibilities and limitations, it is recommended to look at the oneAPI Samples GitHub. This is especially true for direct programming or code migration, such as SYCL instructions.

4 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 one the Spack installations already available in the system, create your own independent environment within this installation[2] or resort to a completely independent personal setup. For more information about how to proceed, we refer to the official documentation and tutorial.

The Ponte Vecchio nodes have their own Spack installation, separated from the global installation adopted for the rest of the system. To set up a proper Spack environment for the Ponte Vecchio nodes, please use the following commands[3]:

Code 5: Spack Ponte Vecchio environment setup

```
export SPACK_DISABLE_LOCAL_CONFIG=true;
export SPACK_ROOT="/shared/gpspack";
source ${SPACK_ROOT}/share/spack/setup-env.sh ;
spack env activate gpqd;
```

[2] Since the /shared filesystem is read-only, it is not possible to create managed environments.
[3] The first line forbids to use a customized setup in the home directory. This precaution reduces the risk of unwanted local side effects, but it is not strictly necessary and often not even desired.

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).

Due to the lack of support or work-in-progress issues, a lot of software still lacks the capability to utilize the hardware. As such, the software stack is currently minimal, but we would be much obliged if you could provide us with requests or information for further applications to be included in this environment, thus helping us in saving time to individual users and in making the hardware more usable.

5 Python

5.1 Load Python 3.11

When using different pieces of software, very often not all versions work fine together. Although other combinations are possible, in this tutorial we will use Python 3.11[4].
After having set up Spack[5], we type:

Code 6: Load pip with python 3.11

```
spack load py-pip@23.1.2
```

This version of pip was built with Spack imposing a Python 3.11 installation as dependence, therefore, when it is loaded, this installation is automatically activated, too. The command “python –version” should now return the answer “3.11.9”.

5.2 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**, containing the path of a directory mounted on a device with sufficient space.
Although it is not guaranteed that, in a shared node, there will be enough room for the needs of all users, the first choice should be a RAM disk, that is automatically created for every user in *\dev\shm*. If the space is not enough and at some point you get the error “No space left on device”, then you should create a temporary directory in *\work\scratch*. Assuming that you have created the directory “tmp_$USER” under *\dev\shm*, the command to let the system know that you have chosen this folder as a temporary directory is:

Code 7: Export the path of a temporary directory and create it

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

5.3 Python modules

Intel provides extensions for popular Python-based frameworks such as PyTorch and TensorFlow. It is strongly recommended to create a virtual environment for your Python modules. In Code 8 you can find the syntax for an environment called “your_environment”.

Code 8: Create virtual enviroment for your python package.

```
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 8), but you need to activate it every time before you use it (second line).
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 for Tensorflow

[4] With Python 3.9, the RHEL 9.4 system default, some basic tests with Tensorflow crashed, while with Python 3.12, that can be loaded after the module intel/2025.0, we did not find Tensorflow versions that could be installed with pip which were also officially supported by the Intel Tensorflow extensions. However, it is possible that now the situation has changed. We will check the status every now and then.
[5] See Code 5

5.4 PyTorch

For PyTorch, we follow the procedure described in `https://pytorch.org/docs/main/notes/get_start_xpu.html`.

With the commands:

Code 9: Install Py-Torch

```
python -m venv pytorch_env
source pytorch_env/bin/activate
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/test/xpu
```

we:

1. create a python virtual environment named *pytorch_env*
2. activate the python virtual environment that we just created
3. install the version of Py-Torch (and its most common packages) for Intel XPUs

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

Code 10: Py-Torch installation quick check

```
import torch
torch.xpu.is_available()
```

which should return "True".

It is recommended to try also the examples listed on `https://pytorch.org/docs/main/notes/get_start_xpu.html#examples`.
At the time of writing these notes, they all worked fine.

5.5 TensorFlow

To install the Tensorflow framework on the Intel GPU Max system, first we set up a python virtual environment, then we install the standard Tensorflow and, finally, the Intel extension for this software:

Code 11: Install Tensorflow

```
python -m venv TF_env
source TF_env/bin/activate
pip install tensorflow==2.15.0
pip install --upgrade intel-extension-for-tensorflow[xpu]
```

Although more recent versions of Tensorflow exist, at the time of writing version 15.0 is explicitly required by the Intel extension.

In order to briefly check that the installation went fine, you can launch python and execute:

Code 12: Py-Tensorflow installation quick check

```
import intel_extension_for_tensorflow as itex
print(itex.__version__)
```

Among the many messages that are printed when importing the module (first command in Code 12), there should be the line: "Intel Extension for Tensorflow* GPU backend is loaded." and, of course, the second command should return the actual version of the Intel Extension for Tensorflow.
Additionally, you can execute the short sample python script in the Tensorflow homepage:

Code 13: Py-Tensorflow basic example script

```
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)
```

6 oneAPI SYCL C++ samples using GPUs

In the website `https://github.com/oneapi-src/oneAPI-samples` Intel provides many examples of code based on the oneAPI framework, including some relatively simple programs that can run also on GPUs. The page `https://oneapi-src.github.io/oneAPI-samples/` provides a basic summary of all the available examples. We tested a small subset of them, but unfortunately non always successfully. We suppose that in most cases the failures might depend on small mismatches between Red Hat 9.4, the linux distribution powering the Lichtenberg II cluster, and Ubuntu 22.04, the linux distribution used by Intel to validate the code, but we cannot rule out that, in some cases, they are due to other problems. Therefore we encourage the users to contact us by opening a ticket whenever they encounter unexpected compilation or execution problems.
Here we report a trivial typical usage example, just to give an idea of how to proceed, referring to the Intel documentation for the other samples.

First of all we load the modules delivering oneAPI and CMake:

Code 14: Load the necessary modules

```
module load intel/2025.0
module load intelmpi
module load cmake
```

Then, of course, we need to download the oneAPI git repository and move into the directory of the code sample that we want to compile and run. The chosen example is a simplified version of a typical scientific problem, with some parameters that can be easily changed.

Code 15: Get the oneAPI repository

```
git clone https://github.com/oneapi-src/oneAPI-samples.git
cd oneAPI-samples/DirectProgramming/C++SYCL/StructuredGrids/particle-diffusion
```

Now we build the executable and, with the last command, we run it:

Code 16: Build the code sample

```
mkdir build
cd build
cmake ..
make
make run
```

The initial part of the output (we omit the rest of it) should be (the device offload time slightly changes from run to run):

```
**Running with default parameters**

Running on: Intel(R) Data Center GPU Max 1550
Device Max Work Group Size: 1024
Device Max EUCount: 512
Number of iterations: 10000
Number of particles: 256
Size of the grid: 22
Random number seed: 777

Device Offload time: 0.0340798 s
```