**Usage of Engaging GPUs for AI Applications**

**Questions related to the document:** Javier Viaña

**Email**: [vianajr@mit.edu](mailto:vianajr@mit.edu)

**Abstract**: This document intends to provide a collection of lessons learned by Javier Viaña to set up correctly the Engaging GPU resources available for MKI affiliates. For more detailed information visit <https://orcd-docs.mit.edu>

1. **Partition Information:**

MKI has purchased several nodes in Engaging. We have two main partitions:

**sched\_mit\_mki\_r8**

**sched\_mit\_mki\_preempt\_r8**

The **sched\_mit\_mki\_r8** partition:

* Has 2 nodes node2000 and node2001, which have a limit of 7 days for computations.
* Each of these nodes has 4 NVIDIA A100 GPUs.
* Is the higher priority partition, the jobs that are sent here will be executed with priority over the jobs sent to **sched\_mit\_mki\_preempt\_r8.**

The **sched\_mit\_mki\_preempt\_r8** partition:

* Has various nodes, which have a limit of 14 days for computations.
* Each of these nodes has 4 NVIDIA A100 GPUs.
* Is the lower priority partition, the jobs that are sent here may be finished suddenly due to “PREEMPTION”. If this happens it would mean that a higher priority job was sent to **sched\_mit\_mki\_r8** and took over the resources that were allocated for the job you sent to **sched\_mit\_mki\_preempt\_r8.**

Each GPU has 32 CPU cores. Since our nodes have 4 GPUs, we have 128 CPU cores per node.

1. **Installation of Virtual Environment and Tensorflow GPU Enabled:**

For those doing AI training and using Tensorflow, remember that Tensorflow is lazy. If there are no GPU resources found by Tensorflow when you are installing it in your virtual environment, it won’t install the GPU version of Tensorflow, which should include **cudatoolkits** and **cudnn**. Furthermore, if you are using the shell of Engaging web portal it won’t be possible for you to install Tensorflow GPU enabled. This is because the Engaging shell works with the Centos 7 system and at MKI we use the Rocky 8. So, instead of using the Engaging web shell, use your own computer’s terminal.

First you will need to ssh into Engaging, we want to use the EOFE10 and not the EOFE7. In my case, my username is vianajr so I run:

ssh vianajr@eofe10.mit.edu

The system will ask me to login with my Kerberos account and the authenticate with the DUO app.

For those that have been trying to install Tensorflow before, had issues already, and want to start fresh you can move the conda and the cache. Careful, only do this if you know that you know you won’t loose any information already installed, since it would basically restart conda and cache from zero:

cd

mv .conda .conda1

mv .cache .cache1

Now, before we can actually start, we first need to allocate some GPU resources, so that when we are installing Tensorflow it recognizes that there exists GPU hardware and installs the Tensorflow GPU enabled for you. We can do this simply with a job of 1 GPU, in the sched\_mit\_mki\_r8 partition and requesting 10GB. That should be enough for the installation. We can do this by:

srun -t 60 --gres=gpu:1 -p sched\_mit\_mki\_r8 -n 2 --mem=10GB --pty bash

Now we have to use the rocky8 system. Very important:

module use /orcd/software/community/001/modulefiles/rocky8

Then let’s check which is the available miniforge in this system:

module av miniforge

The 23.11.0-0 should be available, if not use the one you get in the output of the previous command. Then load this miniforge by doing:

module load miniforge/23.11.0-0

Now we can create a virtual environment:

conda create -n YOUR\_ENVIRONMENT\_NAME

Activate the environment with source:

source activate YOUR\_ENVIRONMENT\_NAME

And install Tensorflow, this can take long:

conda install tensorflow

Once it has finished, then you can check if these lines work, you should see that it mentions there are actual GPU devices:

python -c "import tensorflow as tf; print(tf.config.list\_physical\_devices('GPU'))"

python -c "import tensorflow as tf; print(tf.test.is\_gpu\_available())"

You can also install other packages simply by:

conda install scikit-learn

Or many modules at the same time by simply appending them after:

conda install jupyterlab numpy pandas seaborn scipy

Careful, you do not need to install **cudatoolkit** and **cudnn** after installing Tensorflow.

1. **Usage of Virtual Environment:**

Now that we have the environment created, let’s use it to submit some jobs.

Again, if you have left the terminal, first login:

ssh vianajr@eofe10.mit.edu

Then do the same commands we did before, USE, LOAD, and ACTIVATE:

module use /orcd/software/community/001/modulefiles/rocky8

module load miniforge/23.11.0-0

source activate YOUR\_ENVIRONMENT\_NAME

Now you can send jobs that use the GPU resources.

You can do this in 2 ways:

**Option 1:**

With a BASH file. Your BASH file will specify all the details you want for your job, like the partition where you want your job to be executed, and the resources allocated such as the number of GPUs, the hours and memory requested, etc.

sbatch your\_directoy/your\_bash\_file.sh

**Option 2:**

If you want to run python jobs directly from the terminal then you can simply use first the salloc command, and then run your python file. But note that this won't let you run more commands till the python job is done.

Here are a couple examples of salloc, one using 4 GPUs and the other 1 GPU:

salloc -p sched\_mit\_mki\_preempt\_r8 --mem=0 -N 1 --exclusive --gres=gpu:4

salloc -p sched\_mit\_mki\_preempt\_r8 --mem=0 -N 1 --exclusive --gres=gpu:1

Now run your python file:

python DIRECTORY\_OF\_FILE/MY\_FILE.py

1. **Monitoring GPU Usage:**

When using GPU resources you want to see how your GPUs are being used while your jobs are running. You can do this in 2 ways:

**Option 1 – RECOMMENDED:**

If you intend to submit a job with a BASH script, you will be submitting a line of code like this:

sbatch your\_directory/your\_bash\_file.sh

Once you have submitted your job, you can login in Engaging, and check in the Active Jobs tab if your job is in the queue or actually running.

**A screenshot of a computer

Description automatically generated**

Once the resources are allocated and the job is running you can open the information of the job in the Jobs section of Engaging and it will tell you which node is exactly being used by the job.

**A screenshot of a computer

Description automatically generated**

The job that is shown in this image for example, has node2001 allocated. Let us monitor the usage of the node’s GPUs. First, we need to login inside that node, we can do that through ssh:

ssh node2001

Then we simply run the command to see the resources utilized within this node:

nvidia-smi

This will show a chart like this:

**A screenshot of a computer

Description automatically generated**

In this case I am using the 4 GPUs, but you can see that only around a 24-27% is being used. The task I submitted was the training of an AI model, so if I wanted to increase the % of GPU usage I could simply increase the batch size or increase the number of parameters of the model.

This chart is very useful, as it also tells you in the top right which is the CUDA version used by the supercomputer, which sometimes is important to know if you are installing packages that work only with a certain CUDA version, like Tensorflow with GPU usage enabled. In this case, the CUDA Version installed in the Supercomputer: 12.4.

**Option 2:**

If you are just running a .py file directly through the terminal, and not sending a job with a BASH script, then using an interactive window might be the best option. This requires you to run allocate resources first, i.e., running this:

salloc -p sched\_mit\_mki\_preempt\_r8 --mem=0 -N 1 --exclusive --gres=gpu:4

Then you would be able to run your .py file:

python your\_directory/your\_python\_file.py

And then you can run commands like lscpu, nvidia-smi, or top to see the usage of the GPUs. Note that your .py file should be written so that it actually uses the GPU resources.

1. **Useful Terminal Commands:**

Here are some useful commands to run from the terminal to know the information of the partitions that you have access to:

To check which are the partitions to which you can submit:

groups

To check all the available partitions:

sinfo

To check if the partitions/nodes have GPUs or CPUs:

sinfo -o %f,%G

To check only those that have GPUs:

sinfo -o %f,%G|grep gpu

1. **Useful Information:**

If you are eofe10 then you are in the HEAD NODE, which doesn't have GPUs.

All the resources are in the COMPUTE NODEs, for example, node2000 and node2001 are COMPUTE NODEs.

Note the differences between:

If I am using the HEAD NODE with a virtual environment called javier\_env, then the command line should look like this:

(javier\_env) [vianajr@eofe10 ~]$

If I am using the COMPUTE NODE node2000, then the command line should look like this:

[vianajr@node2000 ~]$

If I am using the COMPUTE NODE node2000, with a virtual environment called javier\_env, then the command line should look like this:

(javier\_env) [vianajr@node2000 ~]$

Below are some other useful commands:

First remember to run the use and load commands:

module use /orcd/software/community/001/modulefiles/rocky8

module load miniforge/23.11.0-0

Now, you can run the following command see all the environments available:

conda env list

If you are already inside an environment and you want to deactivate it:

conda deactivate

If you want to delete an environment:

conda remove --name ENVIRONMENT –all

If you want to add a certain channel for package downloading, like conda-forge.

conda config --add channels conda-forge

1. **Submitting an Array of Jobs:**

If we use the word “array” in the bash script, it would request an array of jobs. Since we have 2 nodes each with 4 GPUs, you could for example:

* Have an array of 8 tasks where you have each GPU working individually, but you need to make them talk to each other by making the code parallelizable
* Have an array of 2 tasks where you have 4 GPUs for each task, but again, you need to make them talk to each other by making the code parallelizable.
* Have no array, and just 1 job in 1 node, and if you want you can use up to 4 GPUs.

Here is an example where we request a single task, with 1 node with 4 GPUs. Since we just have 1 task we can have all the 128 CPUs of the node assigned to the task:

#!/bin/bash

#SBATCH --gres=gpu:4

#SBATCH --cpus-per-task=128

#SBATCH --partition=sched\_mit\_mki\_preempt\_r8

#SBATCH -o odir/myout-%j # Specifies the directory where I want the .out file

Here is an example where we send 8 tasks and each task uses 1 GPU, so we are using all the 8 GPUs available. Note that we have to adjust the cpus-per-task to 32 because we have 8 tasks and each task is running in 1 GPU, and we know that each GPU has 32 CPUs.

#!/bin/bash

#SBATCH --gres=gpu:1

#SBATCH --cpus-per-task=32

#SBATCH --array 0-7

#SBATCH --partition=sched\_mit\_mki\_preempt\_r8

#SBATCH -o odir/myout-%j

If you want an array of 2 tasks using all the resources, then we can have 4 GPUs per task, and 128 CPUs per task. That would utilize all the 8 GPUs available.

#!/bin/bash

#SBATCH --gres=gpu:4

#SBATCH --cpus-per-task=128

#SBATCH --array 0-1

#SBATCH --partition=sched\_mit\_mki\_preempt\_r8

#SBATCH -o odir/myout-%j

1. **Remember for Tensorflow Users:**

Ensure that your Tensorflow code is configured to utilize GPUs. You can do this by creating a Tensorflow session with GPU options explicitly set in your python code. For example as follows:

import tensorflow as tf

# Explicitly allow TensorFlow to use GPU memory as needed

config = tf.compat.v1.ConfigProto()

config.gpu\_options.allow\_growth = True

session = tf.compat.v1.Session(config=config)

# Then wherever you are defining your model, you need to use a mirrored strategy to use all the GPUs:

strategy = tf.distribute.MirroredStrategy()

with strategy.scope():

# Here you define the model, compile it, and then perform the fitting.

# For more information check this website:

<https://www.tensorflow.org/guide/distributed_training>