Python HPC Training

Oscar J. Castro-Lopez and THE UL-HPC Team

July, 2024

University of Luxembourg

Introduction

Introduction to the Workshop

Welcome to the workshop! We will explore techniques to significantly improve the performance of Python code using parallel programming.

Introduction to the Workshop

Welcome to the workshop! We will explore techniques to significantly improve the performance of Python code using parallel programming.

Python is known for its ease of use and versatility, but it may not always be the fastest choice for computationally intensive tasks.

Introduction to the Workshop

Welcome to the workshop! We will explore techniques to significantly improve the performance of Python code using parallel programming.

Python is known for its ease of use and versatility, but it may not always be the fastest choice for computationally intensive tasks.

We'll focus on practical techniques and tools for harnessing the power of parallel computing to boost Python performance.

Key Topics We'll Cover:

1. HPC/Slurm basic knowledge
2. Use Jupyter Lab with a HPC system
3. Using multiple cores with Python packages

Key Topics We'll Cover:

1. HPC/Slurm basic knowledge
2. Use Jupyter Lab with a HPC system
3. Using multiple cores with Python packages

By the end of this workshop, you'll have the knowledge and tools to make your Python applications faster and more efficient using these parallel computing techniques.

Let's get started!

Setup Jupyter Lab with the HPC cluster

Jypyter lab

Jupyter is a flexible, popular literate-computing web application for creating notebooks containing code, equations, visualization, and text. Notebooks are documents that contain both computer code and rich text elements (paragraphs, equations, figures, widgets, links).

This tutorial is based on the tutorial available in the Meluxina technical documentation:

For Linux (also works for MAC):

`https://docs.lxp.lu/hpda/datascience-tools/jupyterlab/Jupyter-Linux/`

For Windows:

`https://docs.lxp.lu/hpda/datascience-tools/jupyterlab/Run-Jupyter-Lab-on-Windows-on-the-local-using-MobaXterm/`

Connecting to the cluster

On a terminal type the following:

```
ssh meluxina
```

Then it would show something like the following prompt:

```
Enter passphrase for key '/home/my_user/.ssh/id_rsa':
```

Input your password and you should be connected to the Meluxina cluster and see the welcome message.

Download the material

On a terminal we are going to clone a GitHub repository that has all the material needed for the training:

```
git clone https://github.com/NCCLUX/python-school-compact.
    git
```

SLURM

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.

On a supercomputer, user applications are run through a job scheduler, also called a batch scheduler or queueing system.

Users can run their applications in two essential ways:

1. batch mode: users submit a script 'launcher' file to SLURM, the commands and applications inside are run by SLURM
2. dev mode (interactive): users get connected by SLURM to a (set of) computing nodes directly and can run their applications interactively

Start Jupyter Lab

1. For **Linux/Max** we are going to launch a **batch job**.
2. For **Windows** we are going to launch an **interactive job**.

Starting Jupyter Lab in Linux

View file jupyter_server_launcher.sh

```
#!/bin/bash -l
#SBATCH --account=p200301           # project account
#SBATCH --job-name=JUPYTER          # Job name
#SBATCH --partition=cpu             # Cluster partition
#SBATCH --qos=short                 # SLURM qos
#SBATCH --time=0-4:00:00            # Time to run the Job(HH:MM:SS)
#SBATCH --nodes=1                   # Number of nodes
#SBATCH --ntasks-per-node=1         # Number of tasks per node
#SBATCH --cpus-per-task=128         # CORES per task

# Load Modules
module load Python
pip install --user --upgrade pip
pip install numpy pandas pymp-pypi numba matplotlib
module load JupyterLab

echo "Connect to meluxina: ssh -L 8888:localhost:8888 ${USER}@login.lxp.lu -p 8822" > notebook_commands.log
echo "Port forwarding: ssh -L 8888:localhost:8888 $(hostname)" >> notebook_commands.log
# Launch jupyter notebook
jupyter lab --no-browser --ip "*" --notebook-dir notebook_dir  --port 8888 --NotebookApp.token='token12345' --NotebookApp.password=''
# Use the following in Windows
#salloc -A p200301 -t 03:00:00 -p cpu -q short -N 1 -c 128
```

Linux Step 1: launch batch job

For future use, you have the flexibility to tailor the script according to your requirements. However, for the sake of this tutorial, it's recommended to leave most settings as they are.

Linux Step 1: launch batch job

For future use, you have the flexibility to tailor the script according to your requirements. However, for the sake of this tutorial, it's recommended to leave most settings as they are.

Now, in a terminal connected to the HPC cluster let's launch the script:

```
sbatch jupyter_server_launcher.sh
```

You can check your jobs with the command `squeue`.

Linux step 2: port forwarding

Once your job is running, we are going to get information of the node we got assigned:

```
cat notebook_commands.log
```

This is going to print something like the following:

```
Connect to meluxina:  ssh -L 8888:localhost:8888 username@login.lxp.lu -p 8822
Port forwarding:  ssh -L 8888:localhost:8888 mel0237
```

You will get a slightly different output, your own user name and a different machine name that was allocated (mel0237).

Linux step 2: port forwarding (cont.)

In a new terminal copy and paste the first line starting from 'ssh...'. For example:

```
ssh -L 8888:localhost:8888 username@login.lxp.lu -p 8822
```

If you encounter public key errors, you need to add the ssh-key to your session.

```
eval ‘ssh-agent -s‘
ssh-add ~/.ssh/my_key # Change this with your key
```

If you don't encounter any errors, after you are connected you must enter the command of the second line starting from 'ssh...'. For example:

```
ssh -L 8888:localhost:8888 mel0237
```

Windows step 1: launch an interactive job

Once you are connected to the cluster with MobaXterm, we are going to start an interactive job with the following command:

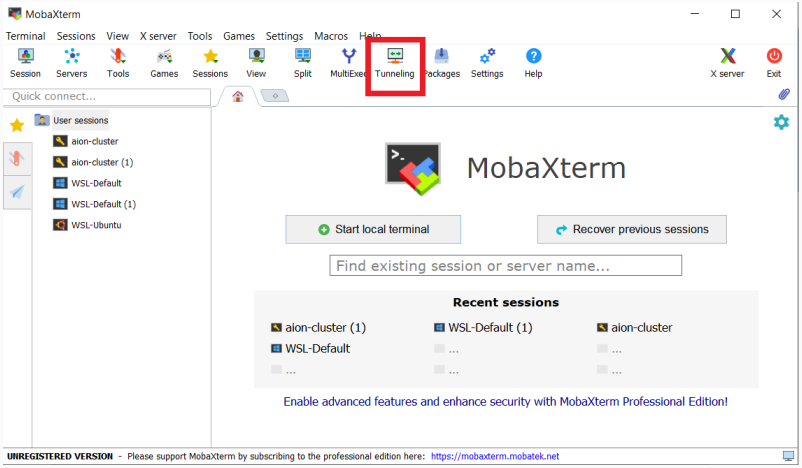
```
salloc -A p200301 -t 03:00:00 -p cpu -q short -N 1 -c 128
```

The following load modules and install libraries(copy-paste from the file `sbatch jupyter_server_launcher.sh`

```
module load Python
pip install --user --upgrade pip
pip install numpy pandas pymp-pypi numba matplotlib
module load JupyterLab
echo "Connect to meluxina: ssh -L 8888:localhost:8888 ${USER
    }@login.lxp.lu -p 8822" > notebook_commands.log
echo "Port forwarding: ssh -L 8888:localhost:8888 $(hostname
    )" >> notebook_commands.log
```

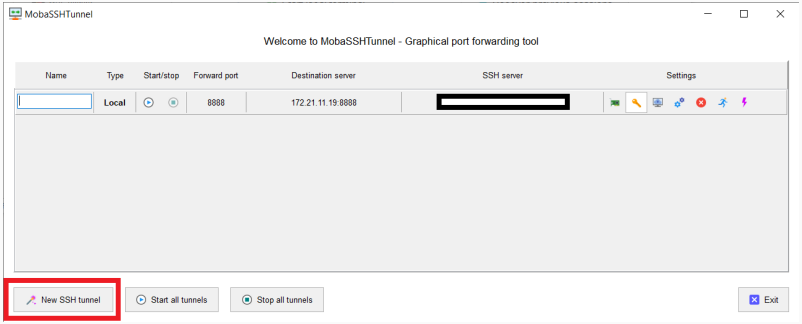
Windows step 2:Forwarding with MobaXterm (pt 1)

1. On MobaXterm windows Open MobaSSHTunnel (click on Tunneling):



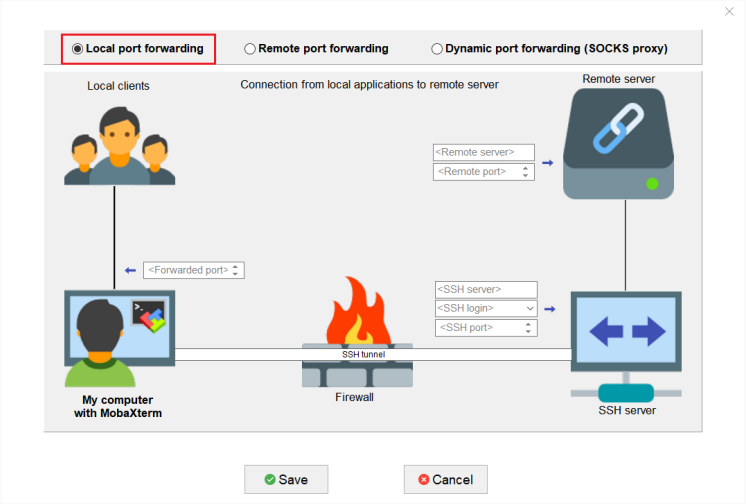
Windows step 2: Forwarding with MobaXterm (pt 2)

2. Create New SSH tunnel:



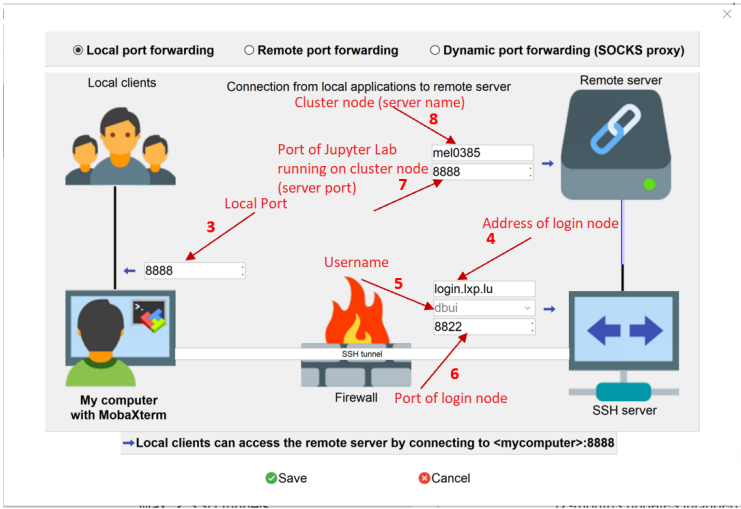
Windows step 2:Forwarding with MobaXterm (pt 3)

3. The following window is going to open:



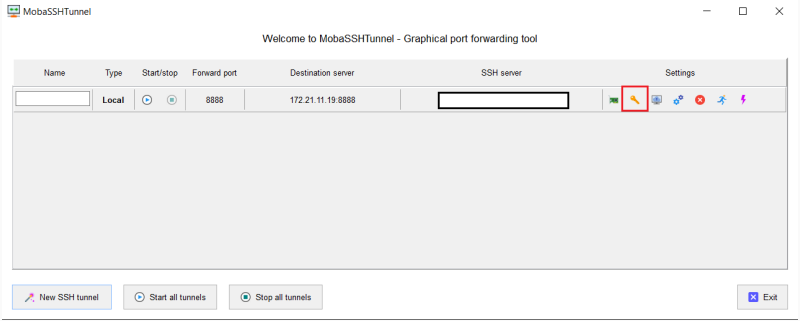
Windows step 2: Forwarding with MobaXterm (pt 4)

4. Type the values taken from the output of notebook_commands.log:



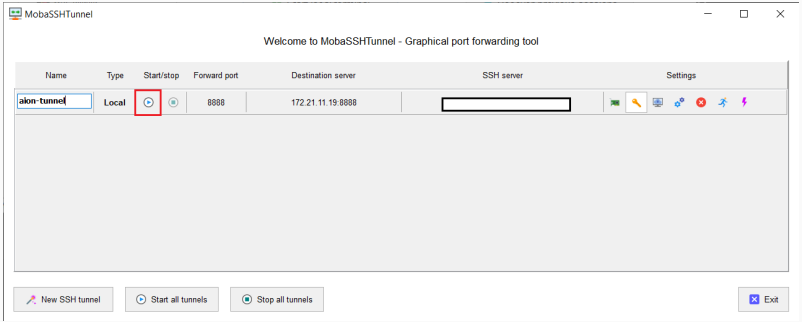
Windows step 2: Forwarding with MobaXterm (pt 5)

5. Add the SSH key:



Windows step 2: Forwarding with MobaXterm (pt 6)

6. Click on the start button (optionally set a name for your forwarding):



Connect to Jupyter Lab (cont.)

To access the Jupyter Lab interface, follow these steps:

1. Open your web browser.
2. Enter the following URL in the address bar: `http://127.0.0.1:8888/`
3. Press Enter to navigate to the provided URL.
4. Jupyter Lab should request a token. Enter your token stated in the command in the provided field.
5. If all is correct, you will be granted access to the Jupyter Lab interface, where you can view your files and begin working.

End

Thank you!

oscar.castro@uni.lu