Python Concurrency

with Threads, Process, Multiprocessing and Futures

Leon Kos[1]

[1]Faculty of Mechanical Engineering
University of Ljubljana

HPC in Data Science: focus on Big Data and AI, June 2023

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Overview

Demonstration and hands-on login1.karolina.it4i.cz

- Running a Jupyter notebook session
- Demonstrating numpy aspects
- Performing computations
- SciPy

Examples for this session:

`https://github.com/kosl/python-training/`

Running a Jupyter notebook session on a LOGIN node

```
# On your local Linux or MacOS machine
ssh it4i-username@login1.karolina.it4i.cz
# Clone the tutorial
git clone https://github.com/kosl/python-training.git
# Load anaconda module on a compute node
module load Anaconda3
# Run the Jupyter lab
jupyter-lab
...
Or copy and paste one of these URLs:
        http://localhost:8888/lab?token=34fa853873fd58aa
```

The port number after the `localhost:` differs according to the free port availability. We need to forward this port to our localhost (laptop) through SSH connection.

Note

Running long computation on a login node is not recommended but can be used for the introductory part although running on a compute node is preferred.

Port forwarding from the login node to the localhost

To establish the port forwarding tunnel from the compute node through the login node to the local machine (laptop) press Enter while the lab process is running to get a clear new line then press ~ (tilda), then C, to get the `ssh>` prompt. Then enter the port forwarding command

```
ssh> -L 8888:localhost:8888
Forwarding port.
```

Note

Port number differs by user. While typing SSH escape command `~C`, to get the `ssh>` prompt, nothing is echoed!

Alternative method to establish a SSH tunnel is to run another terminal connection from the localhost with above connection like

```
ssh -TN -L 8888:localhost:8888 \
  -f it4i-username@login1.karolina.it4i.cz
```

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Running a Jupyter notebook session on a compute node

This method is preferred but can take a while for compute node to be allocated for max 1 hour.

```
# On your local Linux or MacOS machine
ssh it4i-username@login1.karolina.it4i.cz
# Clone the tutorial
git clone https://github.com/kosl/python-training.git
# Acquire your own compute node for an hour
python-training/jupyter-lab.sh
...
Or copy and paste one of these URLs:
        http://localhost:8000/lab?token=c659716f2c10612b
```

The script establishes a new port forwarding tunnel from the compute node to the the login node. User needs only to establish tunnel to local machine (laptop) as described in the previous slide. Key sequence `<Enter>~C` and then `ssh> -L 8000:localhost:8000` in the above port forwarding example.

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

After port forwarding is established at our localhost we can open a connection printed out together with token in any browser. For example

```
http://localhost:8000/lab?token=c659716f2c10612bb6bd6502
```

and immediately the jupyter-lab appears where we select default kernel and first example in file browser on the left, where we open and run step by step `~/python-training/concurrency.ipynb`

Process vs thread for computing

Process

- can run in parallel in Python.
- Uses separate memory space (easy handling, harder communications - IPC)
- Larger memory footprint (usually used in tens - hundreds)

Thread

- can run only concurrently in Python (GIL) - no multicore
- shared memory space (hard management, easy communication)
- lightweight (can be used in hundreds - thousands), in linux 4MB base size

Note

Use of Global Interpreter Lock (GIL) in CPython contributed to widespread use of the language due to its simplicity of implementing libraries with C API without deadlocks.

Threading

- Use the Python threading module to create a multi-threaded application.
- Use the Thread(function, args) to create a new thread.
- Call the start() method of the Thread class to start the thread.
- Call the join() method of the Thread class to wait for the thread to complete in the main thread.
- Global objects still need to be protected between threads.
- Only use threading for I/O bound processing applications.

Note

If the task is doing intensive calculations then multi-threading will not make any speed up.

Multiprocessing

- Multiprocessing module creates new processes (similar as MPI)
- Each created Process() is a clone (fork, spawn) of the master including data
- Function that is started is specified with arguments at creation of the process
- Results can be returned to the master with simple return
- Process creation is expensive system-dependent operation and the size of the problem should be considered
- Python objects can be exchanged through IPC mechanisms. Can be over network.
- Queues, Pipes and Arrays can be used for communication. Size matters!

Futures

Doing concurent and parallel computation easier

- Module concurrent.futures simplifies Thread or Process creation and execution with executors
- Interface simplicity brought to multi-threading and multi-processing with similar functions that allows switching from threads to processes.
- Executor can queue the task to max tasks specified.

Hands-on session

Prepared Jupyter notebooks explain the basics needed for handling data in the following sessions found under `~/python-training/concurrency` directory:

multiprocessing Using (multi) process execution and IPC communication

threading I/O bound threads in Python

futures a higher level interface to push tasks to a background thread without blocking execution of the calling thread, while still being able to retrieve their results when needed

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP