GWDG
AG-C
Hendrik Nolte

Tutorial 1 / April 27, 2022
Practical Course High-Performance Computing / SoSe 2022/23
60 Minutes Total

Learning Objectives

Learn the basics of single node parallelization in Python. In the end one should:

- Manage your own python environments with *Anaconda*
- Describe the difference between threads and processes in Python
- Discuss the general influence, as well as the advantage and disadvantage of the global interpreter lock (GIL)
- Create a parallel Python script which has a GIL
- Create a parallel Python script which prevents the GIL

Tools

- Anaconda
- Python
- Multiprocessing
- Threading

Contents

Working with Anaconda 1: Tutorial (15 min)

Anaconda can be used to create your own, dedicated python environments. *Anaconda* is already installed on the *SCC* and can be immediately used by you.

Steps

1. Load the *anaconda3* module
2. Create a new environment called *pchpc*
3. Activate the new environment

4. Install necessary modules with pip, try to install *numpy* or *pandas*
5. Submit a simple python script with *sbatch*, which only imports *numpy* and *pandas*. You can print a *Done!* afterwards to have some output on yout *stdout*

Hints

- In your run/batch script, you also need to load the *anaconda3* module and activate your environment
- You can load a module with:

```
module load anaconda3
```

- to be able to install python modules with *pip* use:

```
# note that you could use a different name for your environment than 'pip_test'
conda create -n pip_test pip
```

- You can list your different conda environments with:

```
conda env list
```

- You can remove a conda environments with:

```
conda env remove -n pip_test
```

- You can activate a conda environment with:

```
conda activate pip_test
```

- If you get an error activating your environment run:

```
conda init bash
source ~/.bashrc
conda activate pip_test
```

Numerical Integration - Single Threaded 2: Tutorial (15 min)

In this first example we are going to compute a finite numerical integration of the following function:

$$f(x,y) = exp(-0.5(x^2+y^2)) \tag{1}$$

with $x \in [-1,1]$ and $y \in [-1,1]$. In order to do that, we are using a *Monte Carlo* algorithm. The general idea of a *Monte Carlo* integration is, that instead of calculating the integral by evaluating the integrand ($f(x,y)$) at fixed points of a grid, values for x and y are randomly chosen. To get an idea how to compute the following integral, please have a look at fig. 1:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy = \int_{-1}^{1}\int_{-1}^{1} exp(-0.5(x^2+y^2))dxdy \tag{2}$$

Steps

1. Draw two random numbers for x and y

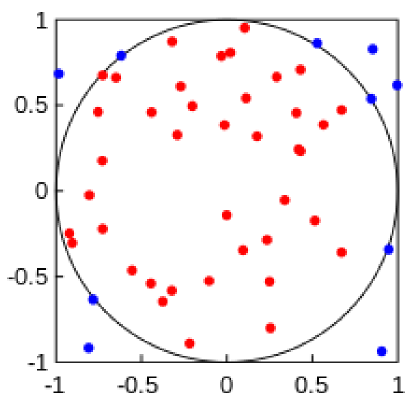

Figure 1: An illustration of Monte Carlo integration. Because the square's area (4) can be easily calculated, the area of the circle ($\pi * 1.0^2$) can be estimated by the ratio (0.8) of the points inside the circle (40) to the total number of points (50), yielding an approximation for the circle's area of $4*0.8 = 3.2 \approx \pi$.

2. Determine the distance of the random 2D point to the origin (0,0). To do that, you can simply calculate $x*x+y*y$
3. Compare the value of your distance with the radius of the unit circle. If it is greater then 1, the point lies outside of your circle, if it is smaller it is inside
4. Keep track of the amount of points inside and outside of your unit circle, as you loop over this method
5. Have a look at fig. 1 and calculate π is described in the caption
6. Write a single threaded function to calculate the integral as shown in fig. 1. Measure the time needed to calculate 10k samples.

Hints

- You can calculate a random number in the range of $x \in [-1, 1]$ with:

```python
import random

x = random.uniform(-1.0,1.0)
```

- You can measure the time a code snippet needs with:

```python
import time

start = time.time()
print("hello")
end = time.time()
print(end - start)
```

Numerical Integration - Multi Threaded 3: Tutorial (15 min)

Building up on the code written before, we are now multi-threading our python script to decrease the overall wallclock time, or to increase the accuracy of our calculations.

Steps

1. Create 10 Threads to calculate your *Monte Carlo* function from the previous Tutorial in parallel

2. Implement a mechanism to gather the results from the individual threads
3. Synchronize all worker threads with the main thread
4. Measure the total wallclock time again and compare it with Tutorial 1
5. Discuss your observations

Hints

- You can multi-thread a python script with:

```python
  import threading

  def my_function(a,b):
    print(str(a*b))

  thread_1 = threading.Thread(target=my_function, args=(2,3))
  thread_2 = threading.Thread(target=my_function, args=(4,5))

  thread_1.start()
  thread_2.start()
```

- When using multi-threads: How can you gather results of the different threads? You could use a result list and each threads insert results in a certain range.
- To print out the final result for π, don't forget to synchronize the threads at the end.

```python
thread_1.join()
thread_2.join()
```

Optional: Numerical Integration - Multi Processing 4: Tutorial (15 min)

To get rid of the global interpreter lock, we are now using multi-processing. Please keep in mind, that now each process has its own distinct memory with now build in process synchronization.

Steps

1. Use multi processing to parallelize your *Monte Carlo* function from the previous Tutorial.
2. Implement a mechanism to gather the results from the individual processes
3. Do you need to synchronize all Processes again?
4. Measure the time and compare with the previous results
5. Describe the differences to multi-threading

Hints

- Multi-Processing is done in Pools. You can create them with:

```python
import multiprocessing as mp

def my_function(a,b):
  return a*b

```

```
pool = mp.Pool(mp.cpu_count())

results = [pool.apply(my_function, args=(a,2)) for a in range(1,100)]

pool.close()
```

Further Reading

- https://docs.anaconda.com/anaconda/user-guide/getting-started/
- https://docs.python.org/3/library/threading.html
- https://docs.python.org/3/library/multiprocessing.html