How to run Python scripts using GNU parallel

Summary

I will provide a tutorial in creating and running a Python script in parallel using multiple cores.

It was intended to be on a high-performance computing cluster using the SLURM workflow management, however I experienced drawbacks in lacking the resources to set this up on the cluster, resulting in doing it on my local machine as a pre-training example of Bash scripts that use more than one core to run.

Prerequisites

- Familiarity with using Ubuntu on Windows
- Familiarity with running Python scripts on Ubuntu
- Familiarity with writing and running Bash scripts on Windows

Process

Set up the environment

On Windows, a Linux distribution that can be used is Ubuntu. You can download Ubuntu from here: https://ubuntu.com/download . After downloading it, the environment needs to be set up, by navigating to where the directory containing all the necessary files is located.

Create the script

The script to run on Ubuntu is written in Bash and contains a Python script that used the same data as my previous tutorials used – csv files with personal information from 17th century historical documents. The Bash script contains a sequence of around 219 numbers/years that represent the values of one command-line argument of the Python script.

In the previous tutorials, I used a for loop to go through the values of that argument, which can be replaced by using GNU parallel. For many values, GNU parallel can enable the Bash script to go through all the argument's values on different cores.

The script

Two Bash scripts were created – one named **files.sh** and one named **parallelscript.sh**

Files.sh script was written to create files (named in the script as **output_$year**) to store the output from each iteration separately.

Parallelscript.sh was written to GNU parallelize the iterations on 2 cores, that my local machine has (however, a cluster has more than 2 cores, and the GNU parallel is more useful for a cluster than for a local machine). For 2 cores, the process is as follows:

The first core and second core run the first two iterations. When one of them finishes the task, another iteration process is allocated to it and so on, until the number of iterations comes to an end. This helps in a quicker running through all the iterations if you have a script with many iterations.

Files.sh script is contained in **parallelscript.sh** to make sure that the outputs go to the files they are assigned to.

Files.sh script' code

#!/usr/bin/bash – for my local machine

#create the files for the python script

year=$1

python3 network_python.py --birth_year $year --gender male > out_$year – send the output to the files

Parallelscript.sh script' code

#!/usr/bin/bash

parallel -j 2 ./files.sh {} ::: `seq 1551 1769` - parallelize the Python script iterations from 1551 to 1769

Run the script

Because it is on my local machine, I am using the **bash filename.sh** method to run the script, as explained in the first of the three tutorials. However, I only run the **parallelscript.sh** script because it also contains the other script.

I then check the time it has taken to run, with the **time** command and discover that it has taken around 2 minutes and 40 seconds on my local machine. A possibility might be that with more than 2 cores on a cluster, it might be faster than on my local machine.

Take-away points

- As mentioned also in the second tutorial discussing the SLUM workflow, care should be taken to wisely allocate resources when running your scripts, be they time, memory space or processors to ensure that you have enough resources to complete the running of your scripts, but also ensure that you do not rob the other users of their resources
- Using the **time** command helps the programmers (the Arts & Humanities researchers here) gauge how much a script takes to run (this is useful for scripts taking longer to finish, such as several hours)