An example of using JASMIN with Python

Eddy Comyn-Platt & Tom August

26 February, 2018

Contents

Introduction

This document and the accompanying code as designed to give you an introduction to show you how to use JASMIN to parallelise your Python code. We will start with some Python code and work though how we can make it parallel. We will then go through how we can get it to run on JASMIN.

What does our script do?

The example script we will work with reads in NETCDF files and calculates some metrics, saving out some plots. This can be found in the file `serial_plot_global_mean_map.py`

```
import numpy as np
import netCDF4 as nc
import sys,os
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
import pdb

import data_info

data_dir = '/work/scratch/tomaug/BSUB_EXAMPLE_python/data/'
out_dir = 'output/'
FILETAG = 'DUMMYGCM/BL_DUMMYGCM.dump.DUMMYYEAR0101.0.nc'
variable = 't_soil'
z_layer = 0
nlayers = 5

GCMs = data_info.GCMs()
#print(GCMs)
nGCMs = len(GCMs)

START_YEAR=1850
END_YEAR=2100
```

```
nYEARS = END_YEAR-START_YEAR+1
plot_years = np.arange(START_YEAR, END_YEAR+1)

grid_file=data_dir+'ancillary/grid_info.nc'
print('Reading grid data from: '+grid_file)
grinf=nc.Dataset(grid_file,'r')
grid_index = grinf.variables['land_index'][:]
lats_2d = grinf.variables['latitude'][:]
lons_2d = grinf.variables['longitude'][:]
grinf.close()

for igcm in range(nGCMs):

    gcm = GCMs[igcm]

    data = []
    for iyear in range(nYEARS):
        year = START_YEAR+iyear
        str_year = str(year)
        infile = data_dir+FILETAG.replace('DUMMYGCM',gcm).replace('DUMMYYEAR',str_year)
        os.system('gunzip '+infile+'.gz')
        print('Reading data from: '+infile)
        inf=nc.Dataset(infile,'r')
        indata = inf.variables[variable][z_layer:z_layer+nlayers,:]
        inf.close()
        os.system('gzip '+infile)

        data.append(np.mean(indata,axis=0))
        del indata

    data = np.array(data)
    mean_data = np.mean(data,axis=0)
    mean_growth_rate = np.mean(data[1:,:]-data[:-1,:],axis=0)
    TS_data   = np.mean(data,axis=1)

    plot_data = np.ma.masked_array(mean_data[grid_index],mask=grid_index.mask)
    plt.imshow(plot_data,origin='bottom')
    plt.colorbar()
    plt.title(variable+' Global Mean ('+str(START_YEAR)+'-'+str(END_YEAR)+')')
    plt.savefig(out_dir+gcm+'_Global_Mean_'+variable+'_Map.png')
    plt.close()

    plot_data = np.ma.masked_array(mean_growth_rate[grid_index],mask=grid_index.mask)
    plt.imshow(plot_data,origin='bottom')
    plt.colorbar()
    plt.title(variable+' Global Mean ('+str(START_YEAR)+'-'+str(END_YEAR)+')')
    plt.savefig(out_dir+gcm+'_Global_MeanGrowth_'+variable+'_Map.png')
    plt.close()

    plt.plot(plot_years,TS_data)
    plt.savefig(out_dir+gcm+'_Global_Mean_'+variable+'_Timeseries.png')
    plt.close()
```

```
    if igcm==1: quit()
```

There are a few important things to pull out of this script. First the `data_dir` parameter tell us where the NetCDF files are stored. The location here is a drive on JASMIN so this won't run locally. Secondly, the script loops through NetCDF files that represent different GCMs (Global Circulation Models). This is done by reading in the names of the GCMs from `data_info.py`, but this could just as easily be a `.csv` or other file type. Thirdly the script progresses through each of the files in turn using a for loop, at each iteration writing out three plots.

- Is this script running the analysis in serial or in parallel?
- Running one GCM takes 5 minutes, there are 34 GCMs, is this a big enough job to warrant the use of JASMIN?
- Could this have been done in a better way?

Move files to JASMIN

You should by now have set yourself up a JASMIN account and be familiar with using MobaXterm to access JASMIN and transfer files. Now use MobaXterm to move all of the files and scripts you have been provided with to your home space of JASMIN. Once you have done this use `cd` to change your working directory so that you are inside the top directory. Once you have done this, typing `ls` should reveal all the files and folders in your working directory which will include the files `serial_plot_global_mean_map.py`, `data_info.py` and others. If you don't see these you have either not copied across all the files correctly or you are not in the correct directory.

Creating a job file

The job file is vital to running your code on JASMIN, it has all the important information such as which queue you job will go to, how much time you need, etc. Here is the job script that I have created for this job (`bsub_parallel.pob`):

```
#!/bin/bash
#BSUB -q short-serial
#BSUB -n 1
#BSUB -W 00:20
#BSUB -J SERIAL_PLOT[1-34]
#BSUB -e %J.e
#BSUB -o %J.o

LOG_FILE=z_logs/${LSB_JOBINDEX}.log
ERR_FILE=z_logs/${LSB_JOBINDEX}.err

echo "./parallel_plot_global_mean_map.py ${LSB_JOBINDEX}" > $LOG_FILE
./parallel_plot_global_mean_map.py ${LSB_JOBINDEX} >> $LOG_FILE 2> $ERR_FILE
```

Let's go through this line by line so that you know what it means

`#!/bin/bash` - this says the file is a bash script, this tells Linux how the file should be run. `#BSUB` - All lines that start with this are instructions to the queue system `-q short-serial` - Submit this job to the `short-serial` queue `-n 1` - each task needs only 1 core to run `-W 00:20` - each task needs a maximum of 20 minutes to run. This is in Days:Hours:Minutes `-J SERIAL_PLOT[1-34]` - The job is going to be called `SERIAL_PLOT` (this can be anything you want). The `[1-34]` means the job will be run 34 times, first with the index 1, then 2, and so on to 34. We will come back to this later. `-o %J.o` - The *log* file will be written to `<job number>.o -e %J.e` - The *error* file will be written to `<job number>.e echo "./parallel_plot_global_mean_map.py ${LSB_JOBINDEX}" > $LOG_FILE` - prints to the log file the path

to the script and the job index `./parallel_plot_global_mean_map.py ${LSB_JOBINDEX} >> $LOG_FILE 2> $ERR_FILE` - Execute the python script `parallel_plot_global_mean_map.py` giving it the index number `${LSB_JOBINDEX}`. Send the standard output to the log file `>> $LOG_FILE` and send the error output to the error file `2> $ERR_FILE` This is quite a lot to take in so take a moment to familiarize yourself with what this files means

How would you change the file to fit these new requirements:

- Your job name is now 'My_analysis'
- Your wall time is 30 hours (can you still use the same queue?)
- Your memory requirement is 8GB

Adapting the script for Jasmin

The GCMs can be parallelised because no one GCM run require output from another. This makes the problem good for being run on JASMIN's cluster (called LOTUS).

Each element in `data_info.py` gives a different parameter for our code to work with. You could also have this as a list of paths to data files, species names, lake co-ordinates, etc. This parameter file can be used in a range of ways to allow you to run lots of code in parallel, more later.

For now let's think about how we can change our Python script so that it works on JASMIN (`parallel_plot_global_mean_map.py`).

```
import numpy as np
import netCDF4 as nc
import sys, os
import matplotlib as mpl
mpl.use('Agg')    # This allows plotting from lotus
import matplotlib.pyplot as plt

import data_info

igcm = int(sys.argv[1])-1
data_dir = '/work/scratch/tomaug/BSUB_EXAMPLE_python/data/'
out_dir = 'output/'
FILETAG = 'DUMMYGCM/BL_DUMMYGCM.dump.DUMMYYEAR0101.0.nc'
variable = 't_soil'
z_layer = 0
nlayers = 5

GCMs = data_info.GCMs()
#print(GCMs)
nGCMs = len(GCMs)

START_YEAR=1850
END_YEAR=2100
nYEARS = END_YEAR-START_YEAR+1
plot_years = np.arange(START_YEAR, END_YEAR+1)

grid_file=data_dir+'ancillary/grid_info.nc'
print('Reading grid data from: '+grid_file)
grinf=nc.Dataset(grid_file,'r')
grid_index = grinf.variables['land_index'][:]
lats_2d = grinf.variables['latitude'][:]
lons_2d = grinf.variables['longitude'][:]
```

```
grinf.close()

gcm = GCMs[igcm]

data = []
for iyear in range(nYEARS):
    year = START_YEAR+iyear
    str_year = str(year)
    infile = data_dir+FILETAG.replace('DUMMYGCM',gcm).replace('DUMMYYEAR',str_year)
    os.system('gunzip '+infile+'.gz')
    print('Reading data from: '+infile)
    inf=nc.Dataset(infile,'r')
    indata = inf.variables[variable][z_layer:z_layer+nlayers,:]
    inf.close()
    os.system('gzip '+infile)

    data.append(np.mean(indata,axis=0))
    del indata

data = np.array(data)
mean_data = np.mean(data,axis=0)
mean_growth_rate = np.mean(data[1:,:]-data[:-1,:],axis=0)
TS_data   = np.mean(data,axis=1)

plot_data = np.ma.masked_array(mean_data[grid_index],mask=grid_index.mask)
plt.imshow(plot_data,origin='bottom')
plt.colorbar()
plt.title(variable+' Global Mean ('+str(START_YEAR)+'-'+str(END_YEAR)+')')
plt.savefig(out_dir+gcm+'_Global_Mean_'+variable+'_Map.png')
plt.close()

plot_data = np.ma.masked_array(mean_growth_rate[grid_index],mask=grid_index.mask)
plt.imshow(plot_data,origin='bottom')
plt.colorbar()
plt.title(variable+' Global Mean ('+str(START_YEAR)+'-'+str(END_YEAR)+')')
plt.savefig(out_dir+gcm+'_Global_MeanGrowth_'+variable+'_Map.png')
plt.close()

plt.plot(plot_years,TS_data)
plt.savefig(out_dir+gcm+'_Global_Mean_'+variable+'_Timeseries.png')
plt.close()
```

This script is essentially identical to the previous version we had. The only change we need to make is to remove the loop and swap it out for this line:

```
igcm = int(sys.argv[1])-1
```

This line takes the index value from the job file (1, 2, . . . , 30) and then proceeds as in the `for` loop to get the required element from the vector of GCM names in `data_info.py`. We use this to get the name of the GCM/NetCDF file we want to work with however you may use it differently in your own work. For example you might instead use a list of species names (running a model for each) or a list of coordinates (to map a list of field sites).

- Can you identify the key new line we have added?
- If this now runs in 3 minutes was it worth it?

Submitting our job

Now that we have our R script ready and the job script written we can run our job. First, make sure your output folders are empty. Second, make sure you are in the directory with the `bsub_parallel.job` file. When here the job can be submitted with `bsub < bsub_parallel.job` at the Linux terminal. This says, submit my job (`bsub_parallel.job`) to the queue system (`bsub`)

```
bsub < pop_sim.job
```

If the job is submitted successfully you will then get a message like:

```
Job <XXXXXXXX> is submitted to queue <short-serial>
```

This long number is the unique identifier for your job and can be used to do various things. A useful one to know is, `bkill -r XXXXXXXXXX`, which will kill your job immediately (good to know if you have made a mistake).

You can monitor the progress of your job by typing `busers`. This command shows a summary of your activity, with columns for the total number of jobs you have on JASMIN (`njobs`), and how many of these are pending (`pend`), versus running (`run`).

If things go well then output will appear in the two output folders. If things go wrong and output does not appear or looks wrong we need to debug.

Hopefully that failed, so now we have an opportunity to go through debugging

Debugging errors

In our job file there were two lines that specified where log and error files would be written. These are the error and log files for the LOTUS queue system, which are different to the error and log files for the Python script.

```
#BSUB -e %J.e
#BSUB -o %J.o
```

These are the first files we want to check to see what has gone wrong. These files will tell us if anything went wrong in the scheduling of the job (i.e. before the python script was run).

Open up your `.e` file and you should find this error:

```
/home/users/tomaug/.lsbatch/1519646910.8668649.17.shell: line 12: z_logs/17.log: No such file or directo
/home/users/tomaug/.lsbatch/1519646910.8668649.17.shell: line 13: z_logs/17.log: No such file or directo
/home/users/tomaug/.lsbatch/1519646910.8668649.20.shell: line 12: z_logs/20.log: No such file or directo
...
```

This saying that the job script asked for python output to be written to the `z_logs` directory but that directory does not exist so it stopped. We can fix this by creating a folder called `z_logs`. Once you have done this run your script again.

This time the scripts should run successfully. However, if there was an issue with the outputs we could look at the console outputs in the `z_logs` folder. It is there that we would find reports of errors during the running of the Python code.

Going that step further

If you have got this far it is time for you to start on your own projects. Hopefully you have brought something with you that you can work on. If you haven't, find someone who has and team up with them. If you don't want to work with anyone else here are some activities you could do with the example we

have been working through. For the answers you will need to look at the JASMIN help documentation (https://help.jasmin.ac.uk/category/107-batch-computing-on-lotus).

- Copy all the output files back to your space on the CEH network
- How would you submit a job so that it only ran **after** another job has completed
- Change our LOTUS queue log file so that it writes to different file for each index, without overwriting
- Run the job again so that it is limited to running only 2 tasks at the same time