**Login to Param**

The easiest way to login to Param is via its IISER internal IP. If you’re not on the IISER network, then login from a terminal on Tahiti (which is connected to the IISER network).

run:

ssh abel@192.168.1.230

You’ll be asked for an authenticator key and a password. Once you provide these, you will reach the login terminal.

I’ve put all of my files in my scratch folder. cd to /scratch/abel

**Using Param**

The login terminal is on a ‘login node’, which is a low-power node.

Simple tasks (only requiring a single thread; for instance, setting up environments and installing packages), can be performed directly on the login terminal.

For any computational tasks, the job scheduler has to be used - Param uses the Slurm job scheduler. The job scheduler passes your script to the computational nodes.

**Working with Slurm**

To run computational jobs, we need to write a job script that provides instructions to Slurm. Below is an example of a job script.

*#!/bin/bash*

*#SBATCH --ntasks-per-node=8*

*#SBATCH --time=96:00:00*

*#SBATCH --job-name=long\_blob2*

*#SBATCH --partition=standard*

*#SBATCH -N 1*

*#SBATCH --exclude=cn007,gpu003*

*source /home/apps/DL-conda/bin/activate*

*conda activate climt\_25*

*cd /scratch/abel*

*python aqua\_run\_blob.py 1*

Brief explanation of the SBATCH variables

‘N’ is the number of nodes requested - you only need 1 for climt

‘ntasks-per-node’ is the number of threads requested

‘time’ is the amount of time requested (in hours). The job will be killed after this time.

‘partition’ is the computational node class you want to use?? Not sure what other partitions can be requested

‘exclude’ lets you avoid certain nodes you think don’t work for some reason

‘Job-name’ - self explanatory

**To submit a job**

sbatch script\_name.sh

**To check job status**

squeue --user=abel

**To cancel job**

scancel <job-id>

**To check node info**

scontol show node <node\_name>

The output from the script is written to a file named slurm-{jobid}.out on the scratch folder (or whichever folder you submitted the job from??)

**Running multiple instances**

One way to run multiple instances of a job is to pass an identifier to the common script (the 1 passed to the aqua\_run\_blob.py script). This distinguishes the various jobs, and is used internally by the script to save the files in different folders. There may be other ways to do this!

For completion, here’s the aqua\_run\_blob.py script. Some parts of this script can be deleted out!

*import climt*

*from sympl import (TimeDifferencingWrapper)*

*import numpy as np*

*from datetime import timedelta*

*import pickle*

*import time*

*import gzip*

*import xarray as xr*

*import zarr*

*import sys*

*key=int(sys.argv[1])*

*pert\_flag=1*

*# Function to perturb spectral surface pressure array*

*def perturb(X):*

*X=np.array(X)*

*N=np.random.uniform(-1,1,np.shape(X[4]))\*np.sqrt(2)\*10\*\*-4*

*X[4][:]=(X[4]+N)[:]*

*#load state from memory*

*def load\_state(state, core, filename):*

*with gzip.open(filename, 'rb') as f:*

*fields, spec = pickle.load(f)*

*if pert\_flag:*

*core.set\_flag(False)*

*perturb(spec)*

*core.\_gfs\_cython.reinit\_spectral\_arrays(spec)*

*state.update(fields)*

*model\_time\_step = timedelta(minutes=20)*

*# Create components*

*convection = climt.EmanuelConvection()*

*boundary=TimeDifferencingWrapper(climt.SimpleBoundaryLayer(scaling\_land=0.7))*

*radiation = climt.GrayLongwaveRadiation()*

*slab\_surface = climt.SlabSurface()*

*optical\_depth = climt.Frierson06LongwaveOpticalDepth(linear\_optical\_depth\_parameter=1, longwave\_optical\_depth\_at\_equator=6)*

*dycore = climt.GFSDynamicalCore(*

*[boundary ,radiation, convection, slab\_surface], number\_of\_damped\_levels=5*

*)*

*grid = climt.get\_grid(nx=128, ny=64)*

*# Create model state*

*my\_state = climt.get\_default\_state([dycore], grid\_state=grid)*

*dycore(my\_state, model\_time\_step)*

*load\_state(my\_state, dycore, 'spinup\_aquaplanet')*

*# Run model for a year after initialisation*

*for i in range(26280):#26280 one year*

*if i==1:*

*dycore.set\_flag(True)*

*diag, my\_state = dycore(my\_state, model\_time\_step)*

*my\_state.update(diag)*

*my\_state['time'] += model\_time\_step*

*# Function to format climate model output*

*def format\_data(state):*

*arr=[]*

*for i in state.keys():*

*if i!='time':*

*if i in ['air\_temperature','air\_pressure','specific\_humidity',*

*'northward\_wind','eastward\_wind','surface\_air\_pressure', 'surface\_temperature',*

*'surface\_upward\_latent\_heat\_flux', 'surface\_upward\_sensible\_heat\_flux',*

*'boundary\_layer\_height', 'divergence\_of\_wind','upwelling\_longwave\_flux\_in\_air',*

*'downwelling\_longwave\_flux\_in\_air']:*

*if i in ['upwelling\_longwave\_flux\_in\_air','downwelling\_longwave\_flux\_in\_air']:*

*arr.append(state[i][0].rename('surface\_'+i).astype('float32'))*

*else:*

*arr.append(state[i].rename(i).astype('float32'))*

*data=xr.merge(arr)*

*data = data.expand\_dims(time=[my\_state['time']])*

*return(data)*

*# store common data before model start*

*arr\_common=[]*

*for i in ['longitude','latitude','area\_type']:*

*arr\_common.append(my\_state[i].rename(i))*

*store = zarr.storage.DirectoryStore("long\_run/common")*

*data\_common=xr.merge(arr\_common)*

*data\_common.to\_zarr(store=store, mode='w')*

*data\_flag=0*

*index=1*

*with gzip.open('obs\_mask', 'rb') as f:*

*obs\_mask = pickle.load(f)*

*def Obs(state):*

*lat = np.radians(my\_state['latitude'].values[:])*

*O=state['air\_temperature'][0].values[:]*

*return((O\*np.cos(lat)\*obs\_mask).sum())/((np.cos(lat)\*obs\_mask).sum())*

*A=[]*

*# Run model for 25 years and store data*

*for i in range(26280\*20):#26280 one year*

*A.append(Obs(my\_state))*

*if (i+1)%18==0:*

*if data\_flag==0:*

*Data=format\_data(my\_state)*

*data\_flag=1*

*else:*

*Data=xr.combine\_by\_coords([Data,format\_data(my\_state)])*

*if (i+1)%26280==0:*

*data\_flag=0*

*compressor = zarr.Blosc(cname="lz4hc", clevel=5, shuffle=True)*

*enc = {x: {"compressor": compressor} for x in Data}*

*store = zarr.storage.DirectoryStore("long\_run\_short/run"+str(key)+"/year"+str(index))*

*(Data.resample(time='1D').mean()).to\_zarr(store=store, encoding=enc, mode='w')*

*index=index+1*

*diag, my\_state = dycore(my\_state, model\_time\_step)*

*my\_state.update(diag)*

*my\_state['time'] += model\_time\_step*