Running the model in production mode: using the queue.

High performance computing (HPC) systems use a queue system to make best use of system resources, and to evenly distribute jobs between different users. I've worked on three HPC systems to date, and each is quite similar in this regard, although sometimes the commands change a bit.

From the computers standpoint, all that matters about your code is:

a) How many processors will you need, in total, to run the model?
b) How many nodes will you need?
c) How many processors per node will you need?
d) How long will your code run for?

In a perfect world, you wouldn't need to worry about (b) or (c); all we want is to run the model with *n* processors. But processors are connected together in clusters, or nodes, that are more tightly bound together. On NYU's HPC, nodes contain up to 20 processors each. (On IBM deep blue machines, you'll find 32 in each node … probably even more now.) So you need to tell it how many nodes you need, and how many processors per node. (If you don't need them all, you can say so, and the spare ones can be used by other users!) The last part is important for scheduling. To efficiently run all jobs, and make sure certain users don't hog all the CPU time, the queue needs know how long your run will take.

Say you'd like to run the spectral model at T42 resolution (42 spherical harmonics) for 1200 days from a dry start. This resolution has 64 latitudes, so we must run it on a number of processors x that divides 64. Say we'd like 16. If processors come in groups of 8 and 12, we need 2 nodes then, and use 8 processors per node. Now many nodes have 20 processors, so we only need one.

How long will this take? Good question. In the beginning, you just have to run it to get an estimate. For these low resolution runs, they go pretty fast, so you can give it, say, 1 hour, and see what happens. The worst case is that it runs out of time; after 1 hours, it will stop. (This is a good thing, for say we made a mistake, and the model tried to run indefinitely! The queue system would eventually put it out of it's misery!)

Edit the create_runscript.csh file in your jobs directory. I assume you've set it up with your name, etc. when running it interactively, so we'll just focus on the important details. Here we'll run the model for 1200 days, writing output every 400 days.

```
#!/bin/csh -f

# Set the run parameters here.  This script will create a job script.
#-------------------------------------------------------------------#
# Define the experiment and NYU/NCAR variables, if applicable

set job_id     = nrelax_t42l20e_control
# simulation name; there should be a corresponding run_parameters
directory

set t_start    = 0  # day to begin (0 for cold start)
set t_end      = 1200  # final day of integration
set dt_atmos   = 1800    # time step of model (seconds)
set delta      = 400    # time for each run (days)

if ($delta == 0) @ delta = $t_end - $t_start

set npes_per_node   = 16  # number of processors per node
set n_nodes         = 1      # number of nodes requested
set wall_clock_max  = 1:00:00   # wall clock max (in
hours:minutes:seconds)
set memory          = 32    # memory required (in GB)


set model_numerics  = spectral  # spectral (for the pseudospectral
model)
                         # fv (for S.J. Lin's finite volume core)
                         # bgrid for bgrid core
```

```
                              # csfv for cubed sphere finite volume core

set model_radiation = nrelax
      # nrelax for Newtonian relaxation
      # gray   for Gray radiation (Frierson 2006)
      # rrtm   for RRTM radiation
```

Execute the script, to produce a job file.

```
>  ./create_runscript.csh

New job file created: job.nrelax_t42l20e_control_d01200
```

Now, take a look at the start of that script. At the top is a cryptic header that seems to be commented out. (Recall that # means comment in a .csh script.)

```
#!/bin/csh -f
#
##SBATCH --nodes=1
#SBATCH --tasks-per-node=16
#SBATCH --time=1:00:00
#SBATCH --mem=32GB
#SBATCH --job-name=nrelax_t42l20e_control_d01200
#SBATCH --mail-type=END
#SBATCH --mail-user=gerber@cims.nyu.edu
#SBATCH --output=/scratch/epg2/job_output/nrelax_t42l20e_control_d00010.out
```

These "sbatch" commands tell the queue the information about your file. The top says I need 1 nodes, 16 nodes per processors, and will consume 1 hours of "wall time." Wall time is how long your run will take, measured by a clock on the wall. (Yes, you probably could have guessed that!)

This is different from the CPU time that your run needs. CPU time = the wall clock time multiplied by the number of processors used. On some systems, the most important metric is the number of nodes you used times the wall clock. (Sometimes nodes can't be shared, so you get charged for using the whole node, whether your code uses it or not. Thus you want to be efficient about how you spread your code between nodes.)

Next, it specified the memory that you need (which we set in our script above). You also see the name of your run and your e-mail (the computer will e-mail you when the run finishes!). The last line is where the standard output log will be written out. (This all the text the comes out of your scripts as it runs — you saw it in the interactive session. They can be useful for finding out what went wrong if the model crashes, but often they can be ignored.)

Next, it sets some environmental variables on the machine.
A bit further down, it tells you how to submit the script:

```
#---------- TO EXECUTE / CHECK ON STATUS --------------#
#
# To submit to the queue:
#   > sbatch job.nrelax_t42l20e_control_d00010
#
# To see jobs in your queue
#   > squeue -u epg2
#
# To cancel a job,
#   > scancel JOBID
# where the JOBID is a number corresponding to the job, and
# listed by the squeue command
#
#------------------------------------------------------#
```

You submit your job by simply typing

```
> sbatch [name of job script]
```

For the case above, it's

```
> sbatch job.nrelax_t42l20e_control_d01200
```

If you have the aliases from my sample.bashrc, you can save keystrokes:

```
> qb job.t42l20e_hs_d01200
```

Tip: make sure you're in the directory where the script is sitting. Otherwise this won't work!

To check in our job, type my shortcut (or squeue -u nyuID)

```
> qs
   JOBID                              NAME    STATE     TIME TIME_LIM NODES
 4936800    nrelax_t42l20e_control_d01200  RUNNING     0:00  1:00:00     1
```

This tells you all your jobs. (squeue will give you more information) Here's what it means, by column,
1) is the queue identifier for the job — this is what you need for cancelling a job.
2) the name of the job (which you gave)
3) state: running (or queued, completed, in error)
4) the time it's been running
5) the total time limit
6) The number of nodes

The alias

```
> qs
```

is a shortcut that provides this custom view of your queue — take a look at your .bashrc file to see what it is in full.

If you need to delete a job (say you recognize a script or the run parameters are incorrect after submitting the job), use the "scancel" command. You need to get queue identify (column 1), from the output you got from qstat. Say I wanted to kill the second job listed above. I'd type

```
> cancel 4936800
```

With hope your job will run smoothly and produce output. You should get an e-mail when it finishes. The next write up will help you interpret the output!

But now that you know how to use the queue system, you can run more advanced models. I've given you four other configurations to start from in:

```
    student_models/run_parameters/
```

and will discuss them briefly here. To run these models, you just switch the simulationID to the appropriate set of parameters below.

How to run GRAM

There are two ways to run the Gray Radiation Aquaplanet Moist GCM developed by Dargan Frierson and co-authors.

1) To use his actual code (or something close), start with this integration set up (that is, this simulationID):

```
gray_t42l25_sbm_control
```

This runs GRAM at T42 resolution with 25 levels. To use this model, you need to specify the appropriate numerics and diabatic scheme in create_runscript.csh.

```
set model_numerics  = spectral

set model_radiation = gray
```

In addition, you my need to lower the time step (I know 600 s works, but try 1800 to start.)

2) Martin set up MiMA to be able to use the gray radiation scheme with a single switch. (Why would you want to do this, you ask? Perhaps you want to keep the same surface conditions as a run with full radiation, but switch the radiation scheme). An example set of parameters to do this is here:

```
mima_gray_t42l40cig_flat
```

Here, you need to tell the code to use MiMA, however:

```
set model_numerics  = spectral

set model_radiation = rrtm
```

The last part says use MiMA (which uses rrtm radiation), but the `namelists` in this set of run parameters has flags to switch from rrtm radiation to gray radiation. (Compare them to the namelists in the MiMA runs to see how!)

How to run MiMA

I've also provided run parameter directories to run MiMA.   For both of these runs, you need to tell create_runscript.csh that you want spectral numerics and rrtm radiation, as with the second version of the gray code above.

```
mima_t42l40cig_flat
```

This runs MiMA with T42 horizontal resolution and 40 vertical layers (spaced according to a formula set up by Chaim Garfinkel, hence the cig) and a flat, uniform lower boundary with a heat capacity equivalent to 10 meters of ocean water.

`mima_t42l40cig_realistic`

This runs MiMA at the same resolution, but in a more realistic configuration which includes:

- realistic topography (albeit at T42 resolution)
- a crude approximation of land sea contrast, where “land” has a lower heat capacity than the “ocean”)
- a purely zonal Q-flux in the tropics to create a “warm pool” in the west Pacific
- the Alexander and Dunkerton (1999) gravity wave parameterization, tuned to give the model a QBO