Running the model in production mode: using the queue.

1) Codes are executed with "run scripts." These are shell script text files that set up the individual runs and execute the code. The scripts will seem a bit overly complicated to use at first; this is so they can be submitted as "jobs" to the queue.

There's a two step process to running the model. I've generated a shell script, "`create_runscript.csh`" that creates the actual run script to run the cores. You can create run scripts for all three cores with this one script by changing the input parameters; on NCAR not all are currently running, but this could easily be fixed.

First we need to understand how the model works. Any run of the GCM requires input files to tell the code the parameter values for the integration (i.e. what is the rotation rate, the resolution, etc.), the output you'd like to have (i.e. zonal wind, surface pressure, etc.), any tracers you'd like included in the run (a tracer is something advected by the flow, like water vapor or pollution), input topography files, and so forth. The parameters governing the simulations are stored in the directory:

```
models/run_parameters/simulationID
```

where simulationID identifies the run you'd like complete. I found it works well to have one place where you put all the input parameters. This way you can always go back and see exactly what you specified for the simulation. (It turns out that you don't have to specify most parameters. If you don't specify them, the model picks the default value -- you may have to look at the fortran source to find out what the default is.)

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?
c) 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 x processors. But processors are connected together in clusters, or nodes, that are more tightly bound together. On IBM deep blue machines, you'll find 32 in each node. Unless you use the "share" queue, you must use whole nodes. [More on this later; unfortunately I have found the share queue to be unreliable with the dynamical cores.] 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.

2) Setting up a run.

Say you'd like to run the spectral model at T42 resolution (42 spherical harmonics) for 1200 days from a cold start. This resolution has 64 latitudes, so we must run it on a number of processors x that divides 64. Say we'd like only 8, for this small test run. This means we should use the share queue. 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 hours, 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 will cut it off. So we edit create_runscript.csh as follows.

```
set job_id      = t42l20e_hs # experiment name
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       = 200        # time for each run (days)
```

Above we specify the simulationID (there must be a directory by this name with the run parameters in models/run_parameters/), the start date, end date (both in days), time step for the model (in seconds) and the increment to run before restarting, delta. (With these settings, the model will run 200 days, then stop, produce output and restart files, and the restart, integrating for another 200 days, and so forth, until it reaches 1200 total days.) If you've already run the model before, and would like to continue the integration, set t_start to the final date of the earlier integraration. (For example, you run the model for 1200 days, and would like to run another 1200. Set t_start = 1200 and t_end = 2400. The script will automatically find the appropriate restart files, assuming that you have not moved the previous model run out of your /ptmp/userID/model_output/ directory.

```
set npes           = 8  # total number of processors to run with
                        # should be  < n_nodes*npes_per_node
set npes_per_node  = 8    # number of processors per node
set n_nodes        = -    # number of nodes requested [not used
on bluefire!]
set wall_clock_max = 1:00  # wall clock max
                           # (at nyu, in hours:minutes:seconds)
                            # at ncar, hours:minutes

set queue_priority = share   # queue setting needed for ncar
set account_number = ??????? # account number needed for ncar
```

Next, we specify the settings for the queue. We want 8 processors, and will take 8 processors for each node. (Note that this is only wise because we're using the share queue, specified by queue_priority.) Lastly, you need a account number, for which you are authorized to compute. Lastly, the settings below here will be changed less often.

```
set model_numerics = spectral  # spectral (for the
                               #     pseudospectral model)
                        # fv (for S.J. Lin's finite volume core)
                        # bgrid for bgrid core
set experiment_type  = iterate
     # 'iterate' to run the model along, sequentially
     # 'life_cycle' to run a life cycle with specified initial
     #              conditions (requires namelist parameters,
     #              and initial_conditions.nc files.)
     # 'ensemble' to a series of ensemble runs

set platform          = ibm  # 'nyu' for NYU Dell cluster
                                # 'tempest' for my machine
                                # 'ibm' for NCAR machines
set user              = gerber
```

The model numerics allows you to use this script to run other cores. This may not work until you've compiled these other cores (finite volume or b-grid. Please ask me if you'd like to use them!) Next, the platform switch -- this script works on different computing systems. For ncar, just set it to ibm. Lastly, what's you're user name -- this is for setting up the directory structures.

To produce the actual script to run the model, you must execute this script:

```
>  ./create_runscript.csh

New job file created: job.t42l20e_hs_d01200
```

It tells you that a job file was created, named after the simulationID and the last date of integration. 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
#
# Set IBM power6 (bluefire) LSF options
```

```
#----------------------------------------------------------------
--------
#BSUB -P  12345678
#BSUB -a  poe
#BSUB -n  8
#BSUB -R "span[ptile=8]"
#BSUB -J  job.output/t42l20e_hs_d01200
#BSUB -o  job.output/t42l20e_hs_d01200.out
#BSUB -e  job.output/t42l20e_hs_d01200.err
#BSUB -q  share
#BSUB -W  1:00
```

These “BSUB” commands tell the queue the information about your file. The top gives the account number to pay for the integration. POE is something machine dependent. Next, -n indicates that I need 8 processors, total, and -R tells the machine that I will use 8 processors per node. (NOTE: unless you are using the shared queue, always request at least 32 processors per node!!!! If you don’t you’re just wasting the others. In general, jobs should be run with a minimum of 32 processors.)
The next lines give the name of the job, and tell you where (relative to the directory the job script sits) the standard text output and error output the of the job will be written. These can be useful when things go wrong. The -q option specifies the queue. “regular” is the standard option, “share” is appropriate when you want a small number of processors, but doesn’t work so well, in my experience. The -W option indicates that the job will consume 1 hour 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.

For charges, however, what really matters is the wall clock time * the number of nodes you are using. (Shared queue is the exception.) Bluefire is built for jobs that run on at least 32 processors -- better yet 64, as it can split them. (Read more on the CISL pages ... I’m not an expert in this.

3) Submitting the job to the queue (running the model!)

A bit further down the job script file, it tells you how to submit the script:

```
#---------- TO EXECUTE / CHECK ON STATUS --------------#
#
# To submit to the queue:
#   bsub < job.t42l20e_hs_d1200
#
# bhist [jobid#]
#     Will summarize the run (once it's finished).
#
# bacct
#    summarizes all of your runs.
#
# bjobs
#    Should tell you about current runs in the queue
#    (and give you job id #'s)
#
# bpeek
#    Let's you look at the standard err and output from a job.
#
#-----------------------------------------------------#
```

You submit your job by simply typing

```
> bsub < [name of job script]
```

For the case above, it's

```
> bsub j< ob.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:

```
> bjobs
```

If you need to delete a job (say you recognize a script or the run parameters are incorrect after submitting the job), use the "bkill" command. You need to get the job number, using bjobs (column 1, I think)

```
> bkill [job number]
```

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!

While the job is running you can see the standard output using the command bpeek.

4) The model ran - where is the output?

With hope you job will execute correctly. The resulting output will be found in a series of directories:

```
/ptmp/userID/model_output/simulationID_d[final date]
```

For this example:

```
/ptmp/userID/model_output/t42l20e_hs_d00200/
/ptmp/userID/model_output/t42l20e_hs_d00400/
```

etc.

Inside each directory are netcdf files with the model output, and the restart files, in a subdirectory RESTART/

```
> ls /ptmp/userId/model_output/t42l20e_hs_d00200/
atmos_average.nc  diag_table   fms.x  input.nml         RESTART
atmos_daily.nc    field_table  INPUT  logfile.0000.out  time_stamp.out
```

The output files from the model are atmos_average.nc and atmos_daily.nc. The first contains time mean values over the full integration. The latter contains daily output, instantaneous values written out once per day.

```
> ls /scratch/nyuId/model_output/n45l24_hs_d00004/RESTART
atmos_model.res      atmosphere.res.nc       spectral_dynamics.res.nc
```

5) How to restart.

Say you have run the model for 1200 days, as described above, but would like to run it more. All you need to do is edit create_runscript.csh. Keep everything the same except the start and end dates. To restart, set the start date to the final date of the previous run, i.e.

```
set t_start    = 1200          # day to begin (0 for cold start)
set t_end      = 2400        # final day of integration
```

Assuming you haven't changed delta from above, this will cause the model to begin integrating at day 1200, run 200 days, restart, and then run until it completes 2400 total days. The restart option assumes that you have left the previous run in your ptmp output directory. All it needs are the restart files. In this specific case, it will look for:

```
/ptmp/userId/model_output/t42l20e_hs_d001200/RESTART
```

and the files inside it that we listed above. It's okay to move the output files (atmos_daily.nc and atmos_average.nc) where ever you'd like.

I have scripts to help organize the data, renaming files so it's more clear where to find the data, etc.

6) Trouble shooting.

Many things can go wrong, probably many more that I could ever anticipate. Here are some common reasons for the script failing in the past.

Errors when you try to execute `./create_runscript.csh`
a) Shell scripts can be very finicky. When you set an option, make sure there is space between the equals signs
correct: `set t_start = 4`
incorrect: `set t_start =4`

Model does not run when you execute the job script (that is, when you type `./job_simulationID_d[end_date]`

a) Make sure that the numerics option matches the type of model you'd like to run. If you specify `n45l24_hs`, this is a finite volume core. If you set the option `model_numerics = spectral`, the model will fail.

b) The script checks to make sure that you haven't already run the model. If you specify a run that will produce a directory that already exists, it will stop before executing. It should produce an error message to this effect.

c) Is this a restart? Are the restart files in the right place? The script should produce an error message that it can't find the restart files.

d) Did you pick the right number of processors? The model will fail in the start up phase if it can't appropriately partition atmosphere between the number of processors that you have specified.