Astro 585 Lab/Homework #7 (Parallel Programming for Distributed Memory Clusters, GPUs & Cloud)

1. In this exercise, you will learn how to run parallel jobs on the Penn State Research Computing Center.

1a. Using these instructions, submit a job to the RCC cluster and have it run the demo program using multiple processors. Make sure it worked. Note that hammer still doesn't have Julia installed, but (most of?) the lionx* cluster do. [If it takes a long time to start a small, short job on the first cluster you try, try submitting to another cluster.]

1b. Write, submit and test a batch job script that performs the necessary initialization and then run the function time_int_normal_pdf_functions() in HW7_Q1_funcs.jl and output the results to a file.

1c. Submit your above batch job script jobs requesting 1,2,4 or 8 processors. (Remember what time you submitted the jobs.) For 2, 4 and 8 try both requesting processors regardless of whether they are on the same node and also requesting all processors to be on the same node. You can either: 1) modify the script several times or 2) use the command line to override PBS parameters like the nodes and job name. Make sure the different jobs don't all overwrite each other's output data.

1d. How long did the different size jobs sit in the queue before starting? (If you use my example scripts and insert your email address where indicated, then you should get an email when your jobs start and finish.)

1e. Collect the results of the previous jobs and make a plot showing run time versus number of cores. Show both the case where all cores were on the same node and the case where the cores could have been spread over different nodes. You can use different colors for the lines/points or separate figures. [Optional: Do this by writing a script that collects the outputs from the various jobs.]

1f. How well does this calculation parallelize over a cluster? Estimate how many processors spread over different nodes you would need for this calculation to be faster than performing it on a single node with 4 cores.

2. In this exercise, you will learn how to perform parallel calculations using a GPU.

2a. Ssh into a machine that has both Julia and CUDA installed. (Currently, that's sagan.astro.psu.edu. I've asked the RCC to install Julia on tesla, and will let you know if/when they succeed. If you don't have an astro account, then this is a good time to find a friend with one for some more pair coding.) Copy the files from my git repository into a working subdirectory.

2b. Look through the cuda code for the GPU kernel in normal_pdf_gpu.cu. Compile the kernel using

```
/usr/local/cuda-5.5/bin/nvcc -ptx normal_pdf_gpu.cu (if on sagan.astro.psu.edu)
```

or

```
module load cuda
/usr/global/cuda/5.5/bin/nvcc -ptx normal_pdf_gpu.cu (if on tesla.rcc.psu.edu)
```

Read through the julia code that initialized CUDA, loads the module with the GPU kernel from the file, unloads the module and shuts down CUDA in HW7_Q2_setup.jl. Then read through HW7_Q2_funcs.jl to see how the int_normal_gpu(...) function that applies this kernel.
Start Julia. Install the CUDA package (with `Pkg.add("CUDA")` and then load both of these files:

```
include("HW7_Q2_setup.jl")
include("HW7_Q2_funcs.jl")
```

Next, run the code for a few different size data sets like

```
ctx = init_cuda()
md = load_functions()
int_normal_gpu(-1.0, 1.0, 128)
int_normal_gpu(-1.0, 1.0, 1024)
int_normal_gpu(-1.0, 1.0, 16*1024)
int_normal_gpu(-1.0, 1.0, 128*1024)
int_normal_gpu(-1.0, 1.0, 1024*1024)
int_normal_gpu(-1.0, 1.0, 16*1024*1024)
unload_functions(md)
close_cuda(ctx)
```

2c. What is the most time consuming part of the calculation for small data sets? What is the most time consuming part of the calculation for large data sets?

2d. How does the total time scale with the data set size for small data sets? For this calculation, how big does the dataset need to be before the wall clock time starts to scale nearly linearly with the problem size?

2e. How would this change if you were able to perform the summation on the GPU, so that you only needed to download one floating point number, rather than an array of values?

2f. [Optional] Implement the summation on the GPU. How much does the performance improve?

3. In this exercise, we’ll return the PSU RCC cluster, but using a more computationally expensive function. Let’s use your most efficiently parallelized version of the eval_model_on_grid_loops(...) function that you wrote for question #2 of Lab #6.

3a. Perform an analysis of run time versus number of cores similar to what you did in problem 1, but for the parallelized eval_model_on_grid_loops(...) function. Make sure that your jobs save the results to files (and that they write to different file names). Consider using a binary file format to help with performance.

3b. Reading from the associated output files, perform some follow-up analysis, such as interactively plotting the results and testing that the output of parallel jobs matches that of the serial job.

3c. Make a plot showing run time versus number of cores. Show both the case where all cores were on the same node and the case where the cores could have been spread over different nodes (either using different colors for the lines/points or separate figures).

3d. How well does this calculation parallelize over a cluster? Estimate how many processors spread over different nodes you would need for this calculation to be faster than performing it on a single node with 4 cores.

4. In this final exercise, you'll learn one simple way to perform embarrassingly parallel calculations in the cloud. Repeat the calculations from question 3, but performing them in the cloud, so you can gain intuition for when to the RCC cluster will provide significantly better performance and when using the cloud makes sense.
One the easy way to get started with cloud computing for your own calculations is to use "Domino" (http://www.dominoup.com/). This service makes it easy for your jobs to run on the Amazon Elastic Compute Cloud (EC2).

4a. You can create a free trial Domino account and follow their quick start tutorial at http://help.dominoup.com/quickStart . (You can try moving on right away, but if you have any difficulties later one, come back to the domino tutorial and make sure that their quick-start project works before debugging something more complicated.)

4b. Once you've created an account and installed the domino software, then you can setup your own new project by

```
domino create astro585lab7q3
cd astro585lab7q3
```

Then, copy runJulia.sh, HW7_Q4_scr.jl, HW6_Q2_planet_populations.jl and HW6_Q2_planet_populations_once.jl (from my github repository) into that directory.
If you had python and R programs named main.py and main.R, you'd run them with

```
domino run main.py
```

or

```
domino run main.R
```

For our class, Domino has begun "beta support" for Julia. In order to submit a job that will use Julia, copy the runJulia.sh script from my git repository into your directory, and submit a job to the domino cloud with a command like:

```
domino run runJulia.sh -F HW7_Q4_scr.jl
```

Navigate to the url where domino is reporting the status of your job.
When it's done, run

```
domino download
```

to transfer the results and any output files to your computer.

4c. Modify the project to use one of your parallelized versions of the eval_model_on_grid(...) function from HW6_Q2. If your new script is HW7_Q4c_scr.jl, then you'll submit the jobs like

```
domino run runJulia.sh -p 4 -F HW7_Q4c_scr.jl
```

Here, the -p options tells julia to use 4 processes, but doesn't affect how many processors domino assigns to the job. For that you have to go to domino's webpage for your project and click on the blue button "View/change project hardware" in the upper right.

4d. Try changing the project hardware setting (via the project webpage) and then resubmitting your job for each of the free, small and medium machine types. How long did your the jobs sit in the queue before starting for each of the machine types? How did the execution time for your function compare? What about the wall clock time (from beginning of job to when it ends)?

4e. Write a script (you could use Julia, python, perl, shell script, etc.) that submits several jobs, each running evaluate_model(...) only once (or a few times if you like). I.e, you're replacing the call to eval_model_on_grid(...) with a script that submits a series of jobs that collectively evaluate the model at each of the points on the grid. Make sure that they save the output in a fashion that doesn't overwrite the output from other jobs and facilitates reading it back in to plot the result of multiple jobs.

4f. Once you verify that your scripts are working, dial up num_stars to 160,000 and increase the resolution of your grid. How long do these jobs take to start/run?

4g. Using files downloaded to your local laptop/workstation, collect the results into a 3-d array like eval_model_on_grid(...) did. Make contour plots showing the minimum "distance" between the simulated "real" data set and the model data sets as a function of each pair of model parameters (i.e., shape, log_scale, eta). (When making contours in shape and log_scale, plot the minimum distance for any value of eta considered.)

4h. What types of applications would using EC2 & domino be good for?
For what types of calculations would the this combination not be a good fit, but the RCC clusters would be a good fit for?
For what types of calculations is neither well suited?