ECE5640: High Performance Computing Spring 2016
Homework #4 Due Wednesday, March 2 at midnight on Blackboard.
Programs should be run on the Discovery cluster.

Submit two files: a pdf file with answers to questions, and , a zip file named <lastname_firstname_hw4.zip> containing code named according to the question numbers, and a bash file. For example the code in question 2 should be named Q2.

1. (5 pts) Run MATLAB from a single node interactive session on the Discovery cluster. Use the "local" profile. How many workers are available? Note: PCT has a 12 worker maximum so use 12 or fewer workers when you run on one node.

2. (5pts) Write a sequential MATLAB program that uses a **`for`** loop to find the Euclidean distance between two vectors, $a[i]$ & $b[i]$, and stores the result in a vector called $c[i]$. Initialize a and b to random floating point values. Turn in your MATLAB code and your run time.

3. (10 pts) Update your MATLAB code from Question 2 to use the **`parfor`** keyword to enact parallelism. Use 10 workers on one node to implement. Turn in your MATLAB code.

4. (10 pts) Update your MATLAB code from Question 2 to use the **`spmd`** keyword to enact parallelism. State briefly how input data elements are partitioned to each lab (worker) & how each lab determines its start & end indices. Use 10 workers on one node to implement. Turn in your MATLAB code.

5. (10pts) Update your MATLAB code from Question 4 to use the **`distributed`** data type to enact parallelism. Use 10 workers on one node to implement. Note that the following built-in MATLAB functions support **`distributed`** vector inputs:
http://www.mathworks.com/help/distcomp/using-matlab-functions-on-codistributed-arrays.html
Turn in your MATLAB code.

6. (10pts) Compare the run time of your MATLAB code from Questions 2, 3, 4, and 5 for vectors of length 10^3, 10^5, and 10^7. Report run times.

7. (10pts) Calculate Euclidean distance using 4 labs (workers) in a pipelined/systolic manner by implementing ring communications in MATLAB. To be explicit,
(a) Have lab #1 subtract vectors $a[i]$ & $b[i]$ and send the result to lab #2.
(b) Have lab #2 square the input and send the result to lab #3.
(c) Have lab #3 sum the input and send the result to lab #4.
(d) Have lab #4 take the square root of the input and send the result back to lab #1.
Turn in your MATLAB code, and report on the run times. In what situations do you envision that this pipelined implementation might be most beneficial?

8. (20pts) Run your parallel MATLAB code from Questions 2, 3, 4, and 5 using MDCS and 10, 20, & 30 nodes on the Discovery cluster. Report run times. Turn in your bash script and your .m file.

9. (20 pts) Update your MATLAB code from Question 2 to use a GPU to enact parallelism.
Note that the following built-in MATLAB functions support **`gpuArray`** vector inputs:
http://www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html

Report your run times. How large does the vector have to be for the GPU to be faster than the CPU? Turn in your MATLAB code, and answer this question for the following two scenarios:
(a) Initialize vectors $a[i]$ & $b[i]$ to random values on the CPU and do the calculation on the GPU.
(b) Initialize vectors $a[i]$ & $b[i]$ to random values on the GPU and do the calculation on the GPU.
What advice would you give a programmer?