CYEN 405 / CSC 452 / CSC 552 – Distributed and Cloud Computing, Fall 2018

Lab #1: Assigned Tues, Sept 18. Due in PDF format on Moodle Tues, Sept 25 at 11:55 pm.

1. (10 pts) 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 variable called *c*. Initialize *a* and *b* to random floating-point values. Report and plot the run times of your MATLAB code for vectors of increasing length, such as 10^3, 10^5, and 10^7. Turn in your MATLAB code.
(**Note**: if the TA cannot copy/paste your code from whatever format you provide it into MATLAB, then you will not get any credit).

2. (20 pts) Update your MATLAB code from Question 1 to use the `parfor` keyword to enact parallelism. Use 1-10 workers on your local machine to implement. Report and plot the run times of your MATLAB code for increasing numbers of workers (e.g. 1-10) and vectors of increasing length, such as 10^3, 10^5, and 10^7. Answer the question, "At which number of workers is the greatest speedup seen, and why?" Turn in your MATLAB code.

3. (20 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 labs on your local machine to implement. Answer the question, "Does this method show speedup, or under which conditions (e.g. for which types of distributed applications) would use of the `spmd` block be targeted instead?" Turn in your MATLAB code.

4. (20 pts) Update your MATLAB code from Question 3 to use the `distributed` data type to enact parallelism. Use 10 workers on one node to implement. Turn in your MATLAB code.
Note that the following built-in MATLAB functions support `distributed` vector inputs:
https://www.mathworks.com/help/distcomp/run-matlab-functions-with-distributed-arrays.html

5. (20 pts) 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. Does this method produce speedup, or in what situations do you envision that this pipelined implementation might be most beneficial?

6.) (10 pts) In preparation for Lab #2, make yourself an XSEDE account and have ***only one*** member of your team register for the upcoming MPI workshop on October 2-3 here:
https://portal.xsede.org/course-calendar/-/training-user/class/748/session/1963/register
Copy the e-mail sent to the person who registered into your HW submission. Using your XSEDE account, remotely login to bridges.psc.edu and copy the login message into your HW submission. Since the next lab will be about MPI, it would be a good idea to attend as much of this workshop as you feasibly can.