Lab 1 Report

I. Introduction:

The purpose of this lab is to experiment with different MATLAB scripts that executed utilizing different computing concepts (sequential, parallel). We also connected to the LONI supercomputer cluster with our own personal allocation to execute code. The code itself was about initializing matrices that contain large varying input sizes (1e3, 1e5, 1e7) and performing calculations of Euclidean distance on said vectors. Scripts that utilized parallel computing required a specified number of workers to be allocated to execution. The runtime of said scripts was recorded. The machine that the runtimes were recorded on (excluding the last script) is a 2-core machine. This report will review the methods and algorithms behind every script, as well as how the experiment was set up. Then, the runtime results and analysis on said results will be reported. This will all be followed by a conclusion that will sum up the information gathered by this lab.

II. Methods and Algorithms:

Fundamentally, all scripts written in this lab perform a Euclidean distance calculation. Euclidean distance is the straight line between two points in an n dimensional space. The formula for calculating Euclidean distance is as follows:

This formula is relatively straightforward to implement; however, the computational complexity of the algorithm is O(n^2). This means that the runtime increases exponentially as the number of dimensions increases. The machine that these scripts were executed on only has a maximum of two cores. Because of this, runtime on any number of dimensions above two were exceptionally long. All scripts (except lab1g) were executed utilizing two dimensions to keep runtimes relatively low.

Question 1’s lab, lab1a, is the sequential implementation of the Euclidean distance formula. The script utilizes a nested FOR-loop to perform this calculation, and has built-in timer functionality (tic and toc) so that runtimes can be measured. Two vectors, a and b, containing randomized numbers, are instantiated using the “randn” function. An array “c” is also allocated to store the output of the Euclidean distance calculations. The nested FOR-loop then finds the Euclidean distance between the a and b vectors and stores it in the c array. After the program is finished, runtime is printed out through the console.

Question 2’s lab, lab1b, enacts parallelism to calculate the Euclidean distance formula. The script checks to see if there is a parallel pool already existing. If there is, it doesn’t create a new one and returns the parallel pool. If there isn’t a parallel pool, it creates a new one. The rest of the program functions the same as in lab1a, except for the outer FOR-loop being a PARFOR. PARFOR is an instruction that takes a certain block of code and distributes it among several workers for computation. In theory, this should be faster than sequentially running a program because multiple workers executing a single block of code seems faster than a single worker executing that same block of code.

Question 3’s lab, lab1c,

Question 4’s lab, lab1d,

Question 5’s lab, lab1f,

Question 6’s lab, lab1g

III. Experiment:

IV. Results:

V. Conclusion: