COMP 380
Lab 4 - Parallel Password Cracking with MPI

You have been diligently working on new and different ways to multiply two square matrices together for your parallel programming course when you notice that a new folder is in your home directory and contains a curious file: ``PLANS FOR WORLD DOMINATION!!!''. Admittedly, random file drops don't seem like a very good way to distribute such plans, but luckily the sender has encrypted the plans using the DES encryption algorithm.

In order to save the world as we know it, you decide to apply your parallel programming skills and decrypt the messages before the world domination deadline. If only you could submit the solutions described below to GitHub Classroom by the deadline, the world will be saved!

You decide that OpenSHMEM is too much trouble and opt for a solution using MPI. Having a supercomputer available to you, you decide to use the `lotus.rhodes.edu` machine to assist you in the code breaking efforts. Luckily you can find documentation on how to use it online and via Canvas.

The information you receive also has a ZIP file with a set of encrypted files and oddly, an encrypted password entry. You suspect that due to the semi-obvious nature of the email, that the password is probably a short one – perhaps five or less characters (printable, ASCII).

There are three components to this assignment. First, you will write a parallel program to search for a password (unique to you) that can be used to decrypt a secret message. Second, you will write a program the decrypts encrypted files and use that to view the contents of your secret message. Lastly, you will use a second password (shared amongst the class), to run experiments measuring the parallel performance of your solution.

Password Cracking

You manage to find a sample program on the Internet (which has been archived in the Github Classroom assignment) that demonstrates how to work with passwords and encryption. A *password* can be used to verify one's identity. How can we check passwords without storing a clear text version of the password on the authentication server? Enter the *one-way hash function*. This is a function that takes a plaintext string and hashes it into a string of seemingly random garbage. This function only works one-way, when you type in your password, the hashed garbage can be compared against a stored version of the hashed garbage. If they match, then the original passwords must be the same. The function is one-way, in that one cannot take the garbage string and work backwards to produce the original string. The program on that you find demonstrates how to generate this hashed/encrypted string using the OpenSSL library. To crack the password you will have to try all possible combinations until your guesses match the hashed garbage string.

Your password program must use the MPI programming API to perform a parallel search of all possible combination of letters, numbers, and symbols, stopping when it has found a string that hashes to the password found in your email. This is the first objective of the lab.

Once you find the plaintext version of the password, you will still need to decrypt the secret message using the DES encryption algorithm. For the second part of the assignment you should construct a sequential program that uses the OpenSSL libraries to decrypt an encrypted file with your newly found password.

Once you have found the password and written a decrypting utility, you should decrypt the file for you in the ZIP file, using the DES algorithm.

Parallel Performance

You notice that one of the files is named "`passwords.des`", which catches your eye. This file recently made headlines in a dump from Alfred Wikileeks Snowman, where it was released with a text file only containing the string: `wDqVEHozzDT1E`. Since this seems important, you decide to attack this file as well.

Since all of this codebreaking involves parallel supercomputers, you decide to write up a short report for your professor in the hopes that he will be impressed with your enthusiasm and parallel programming prowess. Your report contains graphs of execution time, speedup, and efficiency for your attempts to crack the master file `lotus` on the following processor counts: 1, 8, 48, 96, 192, 480, 1056, 1680, 2112.

Quick! To your editor! And your compiler! Save the world before it's too late!