6CS005 Learning Journal - Semester 1 2018/19

Saad motara 1514723

# Table of Contents

[Table of Contents 1](#_Toc527650969)

[1 POSIX Threads 2](#_Toc527650970)

[1.1 Password Cracking 2](#_Toc527650971)

[1.2 Image Processing 2](#_Toc527650972)

[1.3 Linear Regression 2](#_Toc527650973)

[2 CUDA 3](#_Toc527650974)

[2.1 Password Cracking 3](#_Toc527650975)

[2.2 Image Processing 3](#_Toc527650976)

[2.3 Linear Regression 3](#_Toc527650977)

[3 MPI 4](#_Toc527650978)

[3.1 Password Cracking 4](#_Toc527650979)

[3.2 Image Processing 4](#_Toc527650980)

[3.3 Linear Regression 4](#_Toc527650981)

[4 Verbose Repository Log 5](#_Toc527650982)

# POSIX Threads

## Password Cracking

Insert a table of 10 running times and the mean running time.

|  |  |
| --- | --- |
| Solution explored | Time elapsed |
| 67600 | 520.183299341 seconds |
|  | 523.939482563 seconds |
|  | 521.752942919 seconds |
|  | 520.168320766 seconds |
|  | 521.425810217seconds |
|  | 521.277314081 seconds |
|  | 517.597129896 seconds |
|  | 518.582018821 seconds |
|  | 518.487579090 seconds |
|  | 517.716161698 seconds |

Insert a paragraph that hypothesises how long it would take to run if the number of initials were to be increased to 3. Include your calculations.

The average running time to run the program was 520.1130059, then to figure out how many solutions in calculate per second you take the average run time and divide it by the mean running time which would be 130. and then you take that number and multiply it 228488000by the number of solutions that will be available when having 3 uppercase letters. Then you need to take the amount of solutions from the original and multiply it by 26 for to get how many solutions for the 3rd letter, which would be around 1757600. Now you can take the amount of solutions from 3 letters and multiply it by 130 which would be 228488000 seconds.

Explain your results of running your 3 initial password crackers with relation to your earlier hypothesis.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <crypt.h>

#include <time.h>

#include <math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Demonstrates how to crack an encrypted password using a simple

"brute force" algorithm. Works on passwords that consist only of 2 uppercase

letters and a 2 digit integer. Your personalised data set is included in the

code.

Compile with:

cc -o CrackAZ99-With-Data CrackAZ99-With-Data.c -lcrypt

If you want to analyse the results then use the redirection operator to send

output to a file that you can view using an editor or the less utility:

./CrackAZ99-With-Data > results.txt

Dr Kevan Buckley, University of Wolverhampton, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int n\_passwords = 3;

char \*encrypted\_passwords[] = {

//"$6$KB$yPvAP5BWyF7oqOITVloN9mCAVdel.65miUZrEel72LcJy2KQDuYE6xccHS2ycoxqXDzW.lvbtDU5HuZ733K0X0",

//"$6$KB$4dWyDaT7p8xMWfb6.1R4Q7hQMyezwOgLfAcZhdE4QkH3bylH4EmM.gOVhU7m7K9FHfL.kKhzK6bfHqfA.NDb/0",

//"$6$KB$xELPxo7gwBciT/5FN/.kBwgSPG5RZ8TiW8N0hEER1gyYrQ9ErvSfTbGuDl4w9/QyIP5Lu6W25.G.C8jk6VTYL0",

//"$6$KB$YsfzwjTmZTAEUfD/L/uEqVAmfKGAWSa0ZWg8tSPGT4DZnQNBzLT4vXg2nl7Yd2KZrs4dykykxVIZZLob5ijUb/"

"$6$KB$mczNx5tYSqyEwxnZNEfhZ6NMdfjGHyHfLnOu/kKwxC1RE59K/o74x8rCRFX6vXuCDyhLGBsFvk.grhZLEBa9q1",

"$6$KB$P0iw4e1VDQn.W98MT1amNIPSwFjf4H.GFmSzdQKBFGoK8qZIqvaSIPXDB6wcCNK3KJak14qC1qntCm0nJxubJ1",

"$6$KB$p.viTy0M23kTxeFXNDL6gWrT4JmMi8o6X2hyoQpknf5ZvALWtUmK93Tw7LOtHWveUnbY6nJrmGZ7lfX7VY21b0/"

//theses 3 password should be AAA11, BBB22, CCC33

};

/\*\*

Required by lack of standard function in C.

\*/

void substr(char \*dest, char \*src, int start, int length){

memcpy(dest, src + start, length);

\*(dest + length) = '\0';

}

/\*\*

This function can crack the kind of password explained above. All combinations

that are tried are displayed and when the password is found, #, is put at the

start of the line. Note that one of the most time consuming operations that

it performs is the output of intermediate results, so performance experiments

for this kind of program should not include this. i.e. comment out the printfs.

\*/

void crack(char \*salt\_and\_encrypted){

int x, y, z, s; // Loop counters

char salt[7]; // String used in hashing the password. Need space for \0

char plain[7]; // The combination of letters currently being checked

char \*enc; // Pointer to the encrypted password

int count = 0; // The number of combinations explored so far

substr(salt, salt\_and\_encrypted, 0, 6);

for(x='A'; x<='Z'; x++){

for(y='A'; y<='Z'; y++){

for(s='A'; s<='Z'; s++){

for(z=0; z<=99; z++){

sprintf(plain, "%c%c%c%02d", x, y, s, z);

enc = (char \*) crypt(plain, salt);

count++;

if(strcmp(salt\_and\_encrypted, enc) == 0){

printf("#%-8d%s %s\n", count, plain, enc);

} else {

//printf(" %-8d%s %s\n", count, plain, enc);

}

}

}

}

}

printf("%d solutions explored\n", count);

}

int time\_difference(struct timespec \*start, struct timespec \*finish,

long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if(dn < 0 ) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

int main(int argc, char \*argv[]){

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

int i;

for(i=0;i<n\_passwords;i<i++) {

crack(encrypted\_passwords[i]);

}

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,

(time\_elapsed/1.0e9));

return 0;

}

**Results for 3 Uppercase letters and 2 numbers**

|  |  |
| --- | --- |
| Solutions explored 1757600 | Time taken was 10114.850774299 |

**Explanation for 3 uppercase and 2 numbers**

From running the program with 3 uppercase letter and not 2 you can see a that it took longer for the program to run then the original. However, something to consider is I generated 3 passwords for the program that where 3 uppercase and 2 numbers, this resulted in a quicker time than the one I hypothesized as that one was based on the results of the program running 4 encrypted passwords. However, the program did go through 1757600 but took quick half the estimated time which was 228488000.

**Write a paragraph that compares the original results with those of your multithread password cracker**

When running the multithreaded version of the program the time came out to be 6768.17296417, from looking at this result and comparing them to the original and the one with 3 uppercase letters you can see that the time taken when adding a new letter into the program increase with out using any threads attached. But when you add multiple threads you effectively half the time it would take.

## Image Processing

Insert the image displayed by your program

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <malloc.h>

#include <signal.h>

#include <pthread.h>

#include <time.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Displays two grey scale images. On the left is an image that has come from an

image processing pipeline, just after colour thresholding. On the right is

the result of applying an edge detection convolution operator to the left

image. This program performs that convolution.

Things to note:

- A single unsigned char stores a pixel intensity value. 0 is black, 256 is

white.

- The colour mode used is GL\_LUMINANCE. This uses a single number to

represent a pixel's intensity. In this case we want 256 shades of grey,

which is best stored in eight bits, so GL\_UNSIGNED\_BYTE is specified as

the pixel data type.

To compile adapt the code below wo match your filenames:

cc -o ip\_coursework\_055 ip\_coursework\_055.c -lglut -lGL -lm -pthread

./ip\_coursework\_055

Dr Kevan Buckley, University of Wolverhampton, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define width 100

#define height 72

unsigned char image[], results[width \* height];

struct argThread {

unsigned char \*in;

unsigned char \*out;

int start;

int stride;

};

void \*detect\_edges(void \*args) {

int i;

int n\_pixels = width \* height;

struct argThread \*myargs = args;

unsigned char \*in = myargs->in;

unsigned char \*out = myargs->out;

int stride = myargs->stride;

int start = myargs->start;

for(i=start;i<n\_pixels;i= i + stride) {

int x, y; // the pixel of interest

int b, d, f, h; // the pixels adjacent to x,y used for the calculation

int r; // the result of calculate

y = i / width;

x = i - (width \* y);

if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {

results[i] = 0;

} else {

b = i + width;

d = i - 1;

f = i + 1;

h = i - width;

r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)

+ (in[h] \* -1);

if (r > 0) { // if the result is positive this is an edge pixel

out[i] = 255;

} else {

out[i] = 0;

}

}

}

}

void tidy\_and\_exit() {

exit(0);

}

void sigint\_callback(int signal\_number){

printf("\nInterrupt from keyboard\n");

tidy\_and\_exit();

}

static void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glRasterPos4i(-1, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);

glRasterPos4i(0, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);

glFlush();

}

static void key\_pressed(unsigned char key, int x, int y) {

switch(key){

case 27: // escape

tidy\_and\_exit();

break;

default:

printf("\nPress escape to exit\n");

break;

}

}

int time\_difference(struct timespec \*start, struct timespec \*finish,

long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if(dn < 0 ) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

////////////////////////////////////////////

int main(int argc, char \*\*argv) {

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

struct argThread arg1, arg2, arg3, arg4;

signal(SIGINT, sigint\_callback);

pthread\_t t1,t2,t3,t4;

printf("image dimensions %dx%d\n", width, height);

arg1.in = image;

arg1.out = results;

arg2 = arg1;

arg3 = arg1;

arg4 = arg1;

arg1.start = 0;

arg1.stride = 4;

arg2.start = 1;

arg2.stride = 4;

arg3.start = 2;

arg3.stride = 4;

arg4.start = 3;

arg4.stride = 4;

void \*detect\_edges();

pthread\_create(&t1, NULL, detect\_edges, &arg1);

pthread\_create(&t2, NULL, detect\_edges, &arg2);

pthread\_create(&t3, NULL, detect\_edges, &arg3);

pthread\_create(&t4, NULL, detect\_edges, &arg4);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

pthread\_join(t3, NULL);

pthread\_join(t4, NULL);

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,

(time\_elapsed/1.0e9));

glutInit(&argc, argv);

glutInitWindowSize(width \* 2,height);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);

glutCreateWindow("6CS005 Image Progessing Courework");

glutDisplayFunc(display);

glutKeyboardFunc(key\_pressed);

glClearColor(0.0, 1.0, 0.0, 1.0);

glutMainLoop();

tidy\_and\_exit();

return 0;

}

unsigned char image[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

};

|  |  |
| --- | --- |
| Original Test | Time taken |
| Original Test (1) | 0.000002080 |
| Original Test (2) | 0.000002152 |
| Original Test (3) | 0.000002077 |
| Original Test (4) | 0.000002190 |
| Original Test (5) | 0.000002120 |
| Mean running time | 0.000002108 |

|  |  |
| --- | --- |
| multithread Thread test |  |
| Test 1 | 0.000534001 |
| Test 2 | 0.000452514 |
| Test 3 | 0.000431560 |
| Test 4 | 0.000456626 |
| Test 5 | 0.000505026 |
| Mean time | 0.000475945 |

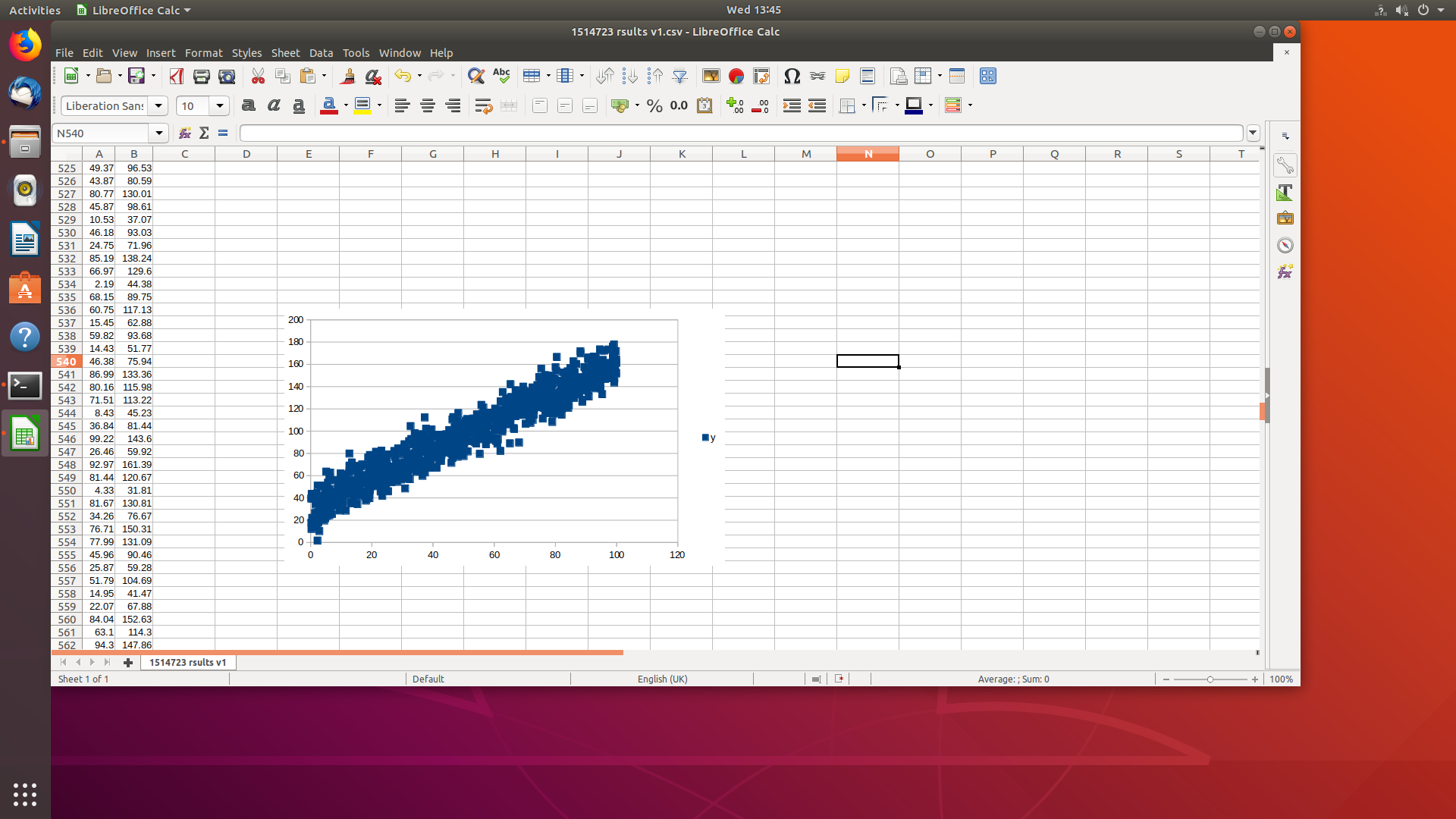
**Insert a table that has columns containing running times for the original program and your multithread version. Mean running times should be included at the bottom of the columns.**

**Insert an explanation of the results presented in the above table.**

From looking at my results the time for the p threaded version should have been faster than the original one as it uses 4 threads to performer the striding method for the image processing. However, what my results show is an increase in time, from the original non-threaded version of the program. The results show an increase in run time for the pthreaded version, this could be based on the fact the code was in efficent in the implementation of the striding method.

## Linear Regression

Insert a scatter plot of your data.



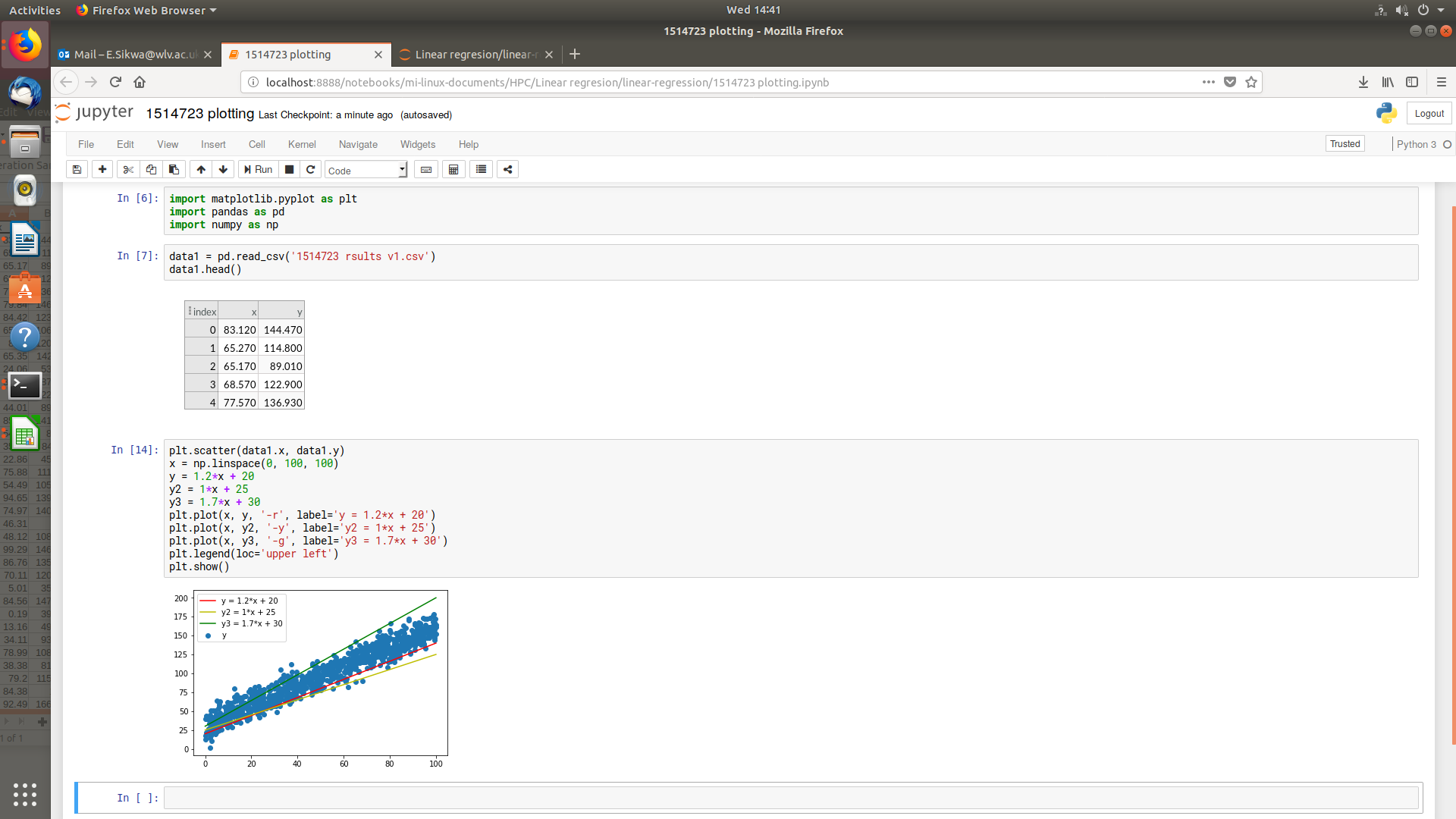
Have 3 guesses at the optimum values for m and c and present them in a graph that overlays your data.

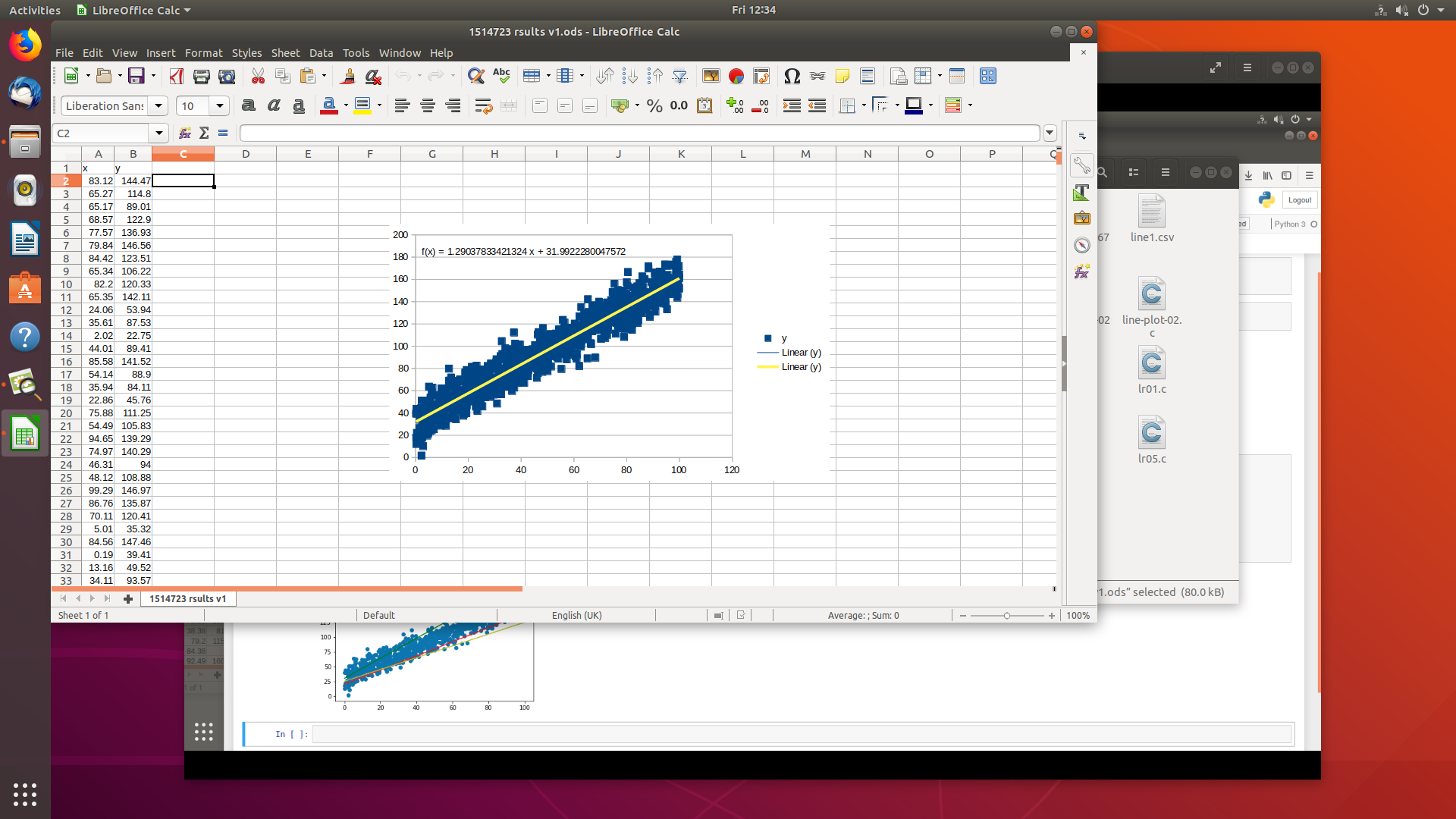
Y= 1.2\*x + 20

Y=1\*x+25

Y= 1.7\*x+30

Insert a graph that presents your data with the solution overlaid.





Insert a comment that compares your guesses with the solution found.

Paste your source code for your multithread linear regression program here.

Insert a table that shows running times for the original and multithread versions.

Write a short analysis of the results.

# CUDA

## Password Cracking

#include <stdio.h>

#include <cuda\_runtime\_api.h>

#include <time.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This program gives an example of a poor way to implement a password cracker

in CUDA C. It is poor because it acheives this with just one thread, which

is obviously not good given the scale of parallelism available to CUDA

programs.

The intentions of this program are:

1) Demonstrate the use of \_\_device\_\_ and \_\_global\_\_ functions

2) Enable a simulation of password cracking in the absence of library

with equivalent functionality to libcrypt. The password to be found

is hardcoded into a function called is\_a\_match.

Compile and run with:

nvcc -o cuda\_crack cuda\_crack.cu

./cuda\_crack

Dr Kevan Buckley, University of Wolverhampton, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This function returns 1 if the attempt at cracking the password is

identical to the plain text password string stored in the program.

Otherwise,it returns 0.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_device\_\_ int is\_a\_match(char \*attempt) {

char plain\_password[] = "gg1111";

char \*a = attempt;

char \*p = plain\_password;

while(\*a == \*p) {

if(\*a == '\0') {

return 1;

}

a++;

p++;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The kernel function assume that there will be only one thread and uses

nested loops to generate all possible passwords and test whether they match

the hidden password.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_global\_\_ void kernel() {

char i, j;

int e;

char password[5];

password[2] = '\0';

for(i='A'; i<='Z'; i++) {

password[0] = i;

for(j='A'; j<='Z'; j++) {

password[1] = j;

for(e=0; e<=9999; e++){

password[2] = e;

if(is\_a\_match(password)) {

printf("password found: %s\n", password);

} else {

//printf("tried: %s\n", password);

}

}

}

}

}

int time\_difference(struct timespec \*start, struct timespec \*finish, long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if (dn < 0) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

int main() {

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

kernel <<<2, 443>>>();

cudaThreadSynchronize();

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elasped was %lldns or %0.9lfs\n", time\_elapsed, (time\_elapsed / 1.0e9));

return 0;

}

Insert a table that shows running times for the original and CUDA versions.

|  |  |
| --- | --- |
| Original | Cuda |
| 520.183299341 | 2.135996805 |
| 523.939482563 | 2.117903748 |
| 521.752942919 | 2.130387672 |
| 518.487579090 | 2.115976956 |
| 521.277314081 | 2.118178623 |

Write a short analysis of the results

After looking at the results of the Cuda version of the password cracking and comparing the results to the original version, the results for the cuda version is more than two times faster than the original version, this is based on the fact that using the GPU allows the program to spawn more threads to perform the task at hand, which is why the there is a huge time disparity between the cuda version of the program.

## Image Processing

Insert a table that shows running times for the original and CUDA versions.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <malloc.h>

#include <signal.h>

#include <cuda\_runtime\_api.h>

//nvcc -o ip\_courswork\_055 ip\_coursework\_055.cu -lglut -lGL

//./ip\_coursework\_055

#define width 100

#define height 72

unsigned char results[width \* height];

unsigned char image[7200] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

};

\_\_global\_\_ void detect\_edges(float \*s\_in, float \*s\_out) {

int i;

int n\_pixels = width \* height;

char \*results;

for(i=0;i<n\_pixels;i++) {

int x, y; // the pixel of interest

int b, d, f, h; // the pixels adjacent to x,y used for the calculation

int r; // the result of calculate

y = i / width;

x = i - (width \* y);

if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {

results[i] = 0;

} else {

b = i + width;

d = i - 1;

f = i + 1;

h = i - width;

r = (s\_in[i] \* 4) + (s\_in[b] \* -1) + (s\_in[d] \* -1) + (s\_in[f] \* -1)

+ (s\_in[h] \* -1);

if (r > 0) { // if the result is positive this is an edge pixel

s\_out[i] = 255;

} else {

s\_out[i] = 0;

}

}

}

}

void tidy\_and\_exit() {

exit(0);

}

void sigint\_callback(int signal\_number){

printf("\nInterrupt from keyboard\n");

tidy\_and\_exit();

}

static void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glRasterPos4i(-1, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);

glRasterPos4i(0, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);

glFlush();

}

static void key\_pressed(unsigned char key, int x, int y) {

switch(key){

case 27: // escape

tidy\_and\_exit();

break;

default:

printf("\nPress escape to exit\n");

break;

}

}

int time\_difference(struct timespec \*start, struct timespec \*finish,

long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if(dn < 0 ) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

int main(int argc, char \*\*argv) {

const int arraySize = sizeof(image);

const int arrayBytes = arraySize \* sizeof(float);

float \*s\_in;

float \*s\_out;

cudaMalloc((void\*\*) &s\_in, arrayBytes);

cudaMalloc((void\*\*) &s\_out, arrayBytes);

detect\_edges <<<5, 50>>>(s\_out, s\_in);

cudaThreadSynchronize();

cudaMemcpy(s\_in, s\_out, arrayBytes, cudaMemcpyDeviceToHost);

cudaFree(s\_in);

cudaFree(s\_out);

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

signal(SIGINT, sigint\_callback);

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,

(time\_elapsed/1.0e9));

printf("image dimensions %dx%d\n", width, height);

//detect\_edges(image, results);

glutInit(&argc, argv);

glutInitWindowSize(width \* 2,height);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);

glutCreateWindow("6CS005 Image Progessing Courework");

glutDisplayFunc(display);

glutKeyboardFunc(key\_pressed);

glClearColor(0.0, 1.0, 0.0, 1.0);

glutMainLoop();

tidy\_and\_exit();

return 0;

}

|  |  |  |
| --- | --- | --- |
| Original | P threaded | Cuba |
| 0.000002120 | 0.000505026 | 0.00000620 |
| 0.000002190 | 0.000456626 | 0.00000747 |
| 0.000002077 | 0.000431560 | 0.00001062 |
| 0.000002152 | 0.000534001 | 0.00000640 |
| 0.000002080 | 0.000534001 | 0.00000610 |

**Write a short analysis of the results**

After running the image processing using Cuda you can see that there is a decrease in the amount of time it takes for the program to run. This Is primary passed on the amount of threads that a GPU can use to help speed up the process. Based on the results from the cuda version of the program you can see that the time is stagnantly faster as the GPU is able to generate more threads for the program to use. However form one of the test results for the cuda version of image processing the time was higher than the average, this could be based on the fact that to many threads or to little thread where used.

## Linear Regression

Paste your source code for your CUDA based linear regression

Insert a table that shows running times for the original and CUDA versions.

Write a short analysis of the results

# MPI

## Password Cracking

Insert a table that shows running times for the original and MPI versions.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <crypt.h>

#include <time.h>

#include <math.h>

#include<mpi.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Demonstrates how to crack an encrypted password using a simple

"brute force" algorithm. Works on passwords that consist only of 2 uppercase

letters and a 2 digit integer. Your personalised data set is included in the

code.

Compile with:

mpicc -o CrackAZ99-With-Data CrackAZ99-With-Data.c -lcrypt

If you want to analyse the results then use the redirection operator to send

output to a file that you can view using an editor or the less utility:

mpiexec ./CrackAZ99-With-Data > results.txt

Dr Kevan Buckley, University of Wolverhampton, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int n\_passwords = 4;

int size;

int rank;

char buffer[4];

char \*encrypted\_passwords[] = {

"$6$KB$yPvAP5BWyF7oqOITVloN9mCAVdel.65miUZrEel72LcJy2KQDuYE6xccHS2ycoxqXDzW.lvbtDU5HuZ733K0X0",

"$6$KB$4dWyDaT7p8xMWfb6.1R4Q7hQMyezwOgLfAcZhdE4QkH3bylH4EmM.gOVhU7m7K9FHfL.kKhzK6bfHqfA.NDb/0",

"$6$KB$xELPxo7gwBciT/5FN/.kBwgSPG5RZ8TiW8N0hEER1gyYrQ9ErvSfTbGuDl4w9/QyIP5Lu6W25.G.C8jk6VTYL0",

"$6$KB$YsfzwjTmZTAEUfD/L/uEqVAmfKGAWSa0ZWg8tSPGT4DZnQNBzLT4vXg2nl7Yd2KZrs4dykykxVIZZLob5ijUb/"

};

/\*\*

Required by lack of standard function in C.

\*/

void substr(char \*dest, char \*src, int start, int length){

memcpy(dest, src + start, length);

\*(dest + length) = '\0';

}

/\*\*

This function can crack the kind of password explained above. All combinations

that are tried are displayed and when the password is found, #, is put at the

start of the line. Note that one of the most time consuming operations that

it performs is the output of intermediate results, so performance experiments

for this kind of program should not include this. i.e. comment out the printfs.

\*/

void crack(char \*salt\_and\_encrypted){

int x, y, z; // Loop counters

char salt[7]; // String used in hashing the password. Need space for \0

char plain[7]; // The combination of letters currently being checked

char \*enc; // Pointer to the encrypted password

int count = 0; // The number of combinations explored so far

char xStart, yStart, xEnd, yEnd;

if (rank == 1) {

xStart = 'A';

yStart = 'A';

xEnd = 'M';

yEnd = 'Z';

}

if (rank == 2) {

xStart = 'N';

yStart = 'A';

xEnd = 'Z';

yEnd = 'Z';

}

substr(salt, salt\_and\_encrypted, 0, 6);

for(x=xStart; x<=xEnd; x++){

for(y=yStart; y<=yEnd; y++){

for(z=0; z<=9999; z++){

sprintf(plain, "%c%c%02d", x, y, z);

enc = (char \*) crypt(plain, salt);

count++;

if(strcmp(salt\_and\_encrypted, enc) == 0){

printf("#%-8d%s %s\n", count, plain, enc);

} else {

//printf(" %-8d%s %s\n", count, plain, enc);

}

}

}

}

printf("%d solutions explored\n", count);

}

int time\_difference(struct timespec \*start, struct timespec \*finish,

long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if(dn < 0 ) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

int main(int argc, char \*argv[]){

MPI\_Status status;

MPI\_Init(NULL, NULL);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

int i;

if (rank == 0) {

MPI\_Finalize();

} else {

for(i=0;i<n\_passwords;i<i++) {

crack(encrypted\_passwords[i]);

}

}

MPI\_Finalize();

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,

(time\_elapsed/1.0e9));

return 0;

}

|  |
| --- |
| MPI TIME |
| 26815.805153773 |

**Write a short analysis of the results**

When running the mpi version of the password cracking, we can see a huge increase in the time it took to run the program, this is based on the fact the program had to add a 2 more numbers to the password, which increased the amount of solution which now become available for the program to run. After running the program, the results showed 3380000 which is the original program went 67600 which is so which added 3 million more solution that the program had to crack, because of this there is a huge increase in the time it takes to run the program. This is becoming more noticeable when you compare the result from the mpi version to the cuda which is extremal faster, as the gpu is able to have access to more threads to run the process.

## Image Processing

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <malloc.h>

#include <signal.h>

#include <mpi.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Displays two grey scale images. On the left is an image that has come from an

image processing pipeline, just after colour thresholding. On the right is

the result of applying an edge detection convolution operator to the left

image. This program performs that convolution.

Things to note:

- A single unsigned char stores a pixel intensity value. 0 is black, 256 is

white.

- The colour mode used is GL\_LUMINANCE. This uses a single number to

represent a pixel's intensity. In this case we want 256 shades of grey,

which is best stored in eight bits, so GL\_UNSIGNED\_BYTE is specified as

the pixel data type.

To compile adapt the code below wo match your filenames:

cc -o ip\_coursework\_055 ip\_coursework\_055.c -lglut -lGL -lm

Dr Kevan Buckley, University of Wolverhampton, 2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define width 100

#define height 72

int size;

int rank;

unsigned char image[], results[width \* height], buffer;

void detect\_edges(unsigned char \*in, unsigned char \*out) {

int i;

int n\_pixels = width \* height;

for(i=0;i<n\_pixels;i++) {

int x, y; // the pixel of interest

int b, d, f, h; // the pixels adjacent to x,y used for the calculation

int r; // the result of calculate

y = i / width;

x = i - (width \* y);

if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {

results[i] = 0;

} else {

b = i + width;

d = i - 1;

f = i + 1;

h = i - width;

r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)

+ (in[h] \* -1);

if (r > 0) { // if the result is positive this is an edge pixel

out[i] = 255;

} else {

out[i] = 0;

}

}

}

}

void tidy\_and\_exit() {

exit(0);

}

void sigint\_callback(int signal\_number){

printf("\nInterrupt from keyboard\n");

tidy\_and\_exit();

}

static void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glRasterPos4i(-1, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);

glRasterPos4i(0, -1, 0, 1);

glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);

glFlush();

}

static void key\_pressed(unsigned char key, int x, int y) {

switch(key){

case 27: // escape

tidy\_and\_exit();

break;

default:

printf("\nPress escape to exit\n");

break;

}

}

int time\_difference(struct timespec \*start, struct timespec \*finish,

long long int \*difference) {

long long int ds = finish->tv\_sec - start->tv\_sec;

long long int dn = finish->tv\_nsec - start->tv\_nsec;

if(dn < 0 ) {

ds--;

dn += 1000000000;

}

\*difference = ds \* 1000000000 + dn;

return !(\*difference > 0);

}

int main(int argc, char \*\*argv) {

MPI\_Status status;

MPI\_Inti(NULL, NULL);

MPI\_Comm\_Size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_Size(MPI\_COMM\_WORLD, &size);

struct timespec start, finish;

long long int time\_elapsed;

clock\_gettime(CLOCK\_MONOTONIC, &start);

signal(SIGINT, sigint\_callback);

detect\_edges(image, results);

clock\_gettime(CLOCK\_MONOTONIC, &finish);

time\_difference(&start, &finish, &time\_elapsed);

printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,

(time\_elapsed/1.0e9));

printf("image dimensions %dx%d\n", width, height);

detect\_edges(image, results);

if (rank ==0) {

MPI\_Recv(buffer, width \* height, MPI\_BYTE, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

int n\_pixel = width \* height;

for(int 1 = status.MPI\_SOURCEl i < n\_pixels; i = i +size){

} else {

MPI\_Send(results, width \* height, MPI\_BYTE, 0, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

glutInit(&argc, argv);

glutInitWindowSize(width \* 2,height);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);

glutCreateWindow("6CS005 Image Progessing Courework");

glutDisplayFunc(display);

glutKeyboardFunc(key\_pressed);

glClearColor(0.0, 1.0, 0.0, 1.0);

glutMainLoop();

tidy\_and\_exit();

return 0;

}

unsigned char image[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,

0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,

255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,

0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,

255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,

255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,

0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,

255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,

255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,

255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

};

Insert a table that shows running times for the original and MPI versions.

|  |  |
| --- | --- |
| Original Test | Time taken |
| Original Test (1) | 0.000002080 |
| Original Test (2) | 0.000002152 |
| Original Test (3) | 0.000002077 |
| Original Test (4) | 0.000002190 |
| Original Test (5) | 0.000002120 |
| Mean running time | 0.000002108 |

|  |  |
| --- | --- |
| MPI Test | Time taken |
| MPI Test (1) | 0.001631862 |
| MPI Test (2) | 0.001631862 |
| MPI Test (3) | 0.001631862 |
| MPI Test (4) | 0.001631862 |
| MPI Test (5) | 0.001631862 |
| Mean running time |  |

Write a short analysis of the results

From looking at my results of the program

The results of the MPI version of image processing shows an increase time in the run time, this has been seen in the original pthreaded version of the program. This could again be based on a bad version of the porgam or the fact that the program becomes slower as striding and mpi version does not make the most efficient uses of the pixels that it is processing.

## Linear Regression

Paste your source code for your MPI based linear regression

Insert a table that shows running times for the original and MPI versions.

Write a short analysis of the results

# Verbose Repository Log

Paste your verbose format repository log here. With subversion this can be achieved by the following:

svn update

svn –v log > log.txt

gedit log.txt

Then select, copy and paste the text here

<https://github.com/SaadMotara/HPC/tree/Orginial> github link