6CS005 Learning Journal - Semester 1 2019/20

Bishal Baba Maka (1930143)

# 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)

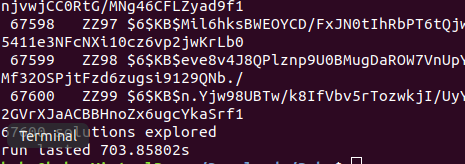
[3.3 Linear Regression 4](#_Toc527650981)

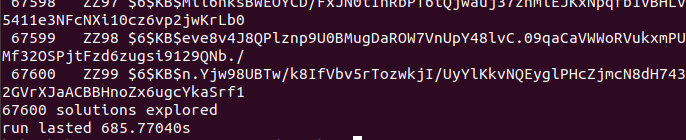
[4 Verbose Repository Log 5](#_Toc527650982)

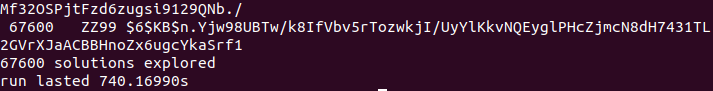
# POSIX Threads

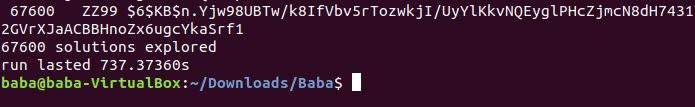
## Password Cracking

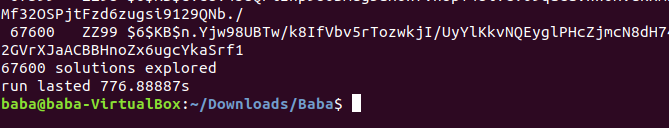
Insert a table of 10 running times and the mean running time.

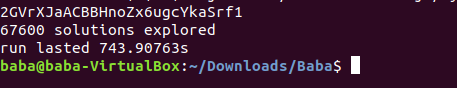


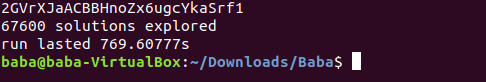


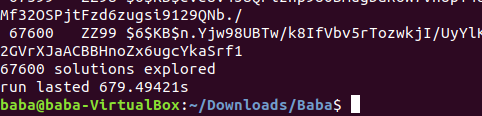


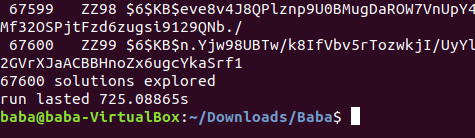


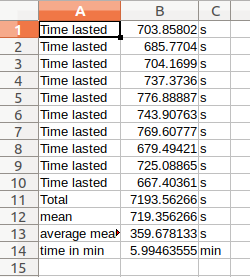
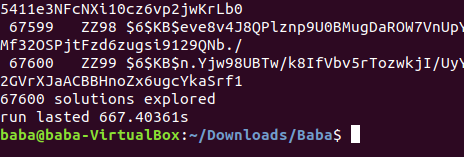












Insert a table of 10 running times and the mean running time.

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.

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <crypt.h>  #include <time.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 Crack-1c Crack-1c.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:  ./Crack-1c > 1c-results.txt  Dr Kevan Buckley, University of Wolverhampton, 2018  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int n\_passwords = 4;  char \*encrypted\_passwords[] = {  "$6$KB$SBwmEVvdlRwwjM30ipUQUNGjTb2zq34CeCk4WfsfLi5Jjj.P5d2oUDMe1qNiy/099nbor7NpnFNygY0ukmJmF0",  "$6$KB$owCbMrAHgYYHcZOJ7YqS6C14KoTNOVoxMml7PYCAyHFW6wkTiPSLta3/A1JY455BzslsWlgzORTlPnq6/BIeA.",  "$6$KB$D19OAN3XSfvODh4G8a4DAeBp.4/ajQ/Ch4VdZWs3uWTzhsL296tJu7NxkupixxPObtVc6fQqojIkkIapRtKSy.",  "$6$KB$r9xIoaj3bEoJnEY9O3ku6tKrIb2fISbbU7rgxAG8QVqmtSUMXOPiqdL2W8Nae3YkY4AMjBJAKrXFvcrmzdWJo/"  };  /\*\*  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, s, 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  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[]){  int i;  struct timespec start, finish;  long long int difference;  int account =0;  clock\_gettime(CLOCK\_MONOTONIC, &start);    for(i=0;i<n\_passwords;i<i++) {  crack(encrypted\_passwords[i]);  }  printf(" solutions explored\n");  clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &difference);  printf("run lasted %9.5lfs\n", difference/1000000000.0);  return 0;  } |

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

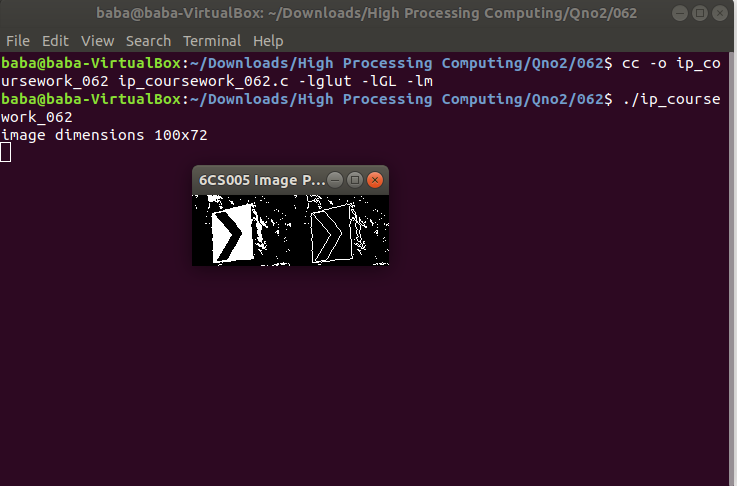
|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <crypt.h>  #include <time.h>  #include <pthread.h>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*  Demonstrates 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 Thread-1e Thread-1e.c -lcrypt -pthread  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:  ./Thread-1e > 1e-Thread\_results.txt  Dr Kevan Buckley, University of Wolverhampton, 2018  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*/  int n\_passwords = 4;  char \*encrypted\_passwords[] = {  "$6$KB$36REBSFzq1/dui1N6SgVpJNWbH7.H09nEmDF84hZhfeauI13ShDyaMT/d1tPMVh35NJKKkQrT6OIYDS9VLrIY0",  "$6$KB$TShKJUQHJ/epFrqd8PVtnjB9lsjrRx6Qwv6fY/9ZOr0NHkH3sFrkc0IZlE8ricgOeQjFu56VSq9.BV2kNSpAg.",  "$6$KB$ielcaeww2oJY5cJuYcgdrtKXuWecpbQLcbvNvI32cdGc1SW17x8csdrpeeUN/Pk8i/aXDLw1Kv4ON/obBDmcT1",  "$6$KB$V51Vk5oaq9jlHAOGUICSX1DRt6gOTPS2N9rx8KI9UPCJtcpdSHV5rmY3c9/B5Ow1eQ62.B0x7.RIlljC9ujU71"  };  /\*\*  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\_bbm()  {  int i;  pthread\_t t1, t2;  void \*kernel\_function\_1();  void \*kernel\_function\_2();  for(i=0;i<n\_passwords;i<i++) {      pthread\_create(&t1, NULL,kernel\_function\_1, encrypted\_passwords[i]);  pthread\_create(&t2, NULL,kernel\_function\_2, encrypted\_passwords[i]);  pthread\_join(t1, NULL);  pthread\_join(t2, NULL);  }  }  void \*kernel\_function\_1(char \*salt\_and\_encrypted){  int d, e, p; // Loop counters  char salt[7]; // String used in hahttps://www.youtube.com/watch?v=L8yJjIGleMwshing the password. Need space  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(d='A'; d<='Z'; d++){  for(e='A'; e<='Z'; e++){  for(p=0; p<=99; p++){  sprintf(plain, "%c%c%02d", d,e,p);  enc = (char \*) crypt(plain, salt);  count++;  if(strcmp(salt\_and\_encrypted, enc) == 0){  printf("#%-8d%s %s\n", count, plain, enc);  }  }  }  }  printf("%d solutions explored\n", count);  }  void \*kernel\_function\_2(char \*salt\_and\_encrypted){  int b, m, c; // Loop counters  char salt[7]; // String used in hahttps://www.youtube.com/watch?v=L8yJjIGleMwshing the password. Need space  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(b='A'; b<='Z'; b++){  for(m='A'; m<='Z'; m++){  for(c=0; c<=99; c++){  sprintf(plain, "%c%c%02d", b,m,c);  enc = (char \*) crypt(plain, salt);  count++;  if(strcmp(salt\_and\_encrypted, enc) == 0){  printf("#%-8d%s %s\n", count, plain, enc);  }  }  }  }  printf("%d solutions explored\n", count);  }  //Calculating time  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);      crack\_bbm();    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;  } |

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

Paste your source code for a 3 initial password cracker here. The code should be neatly indented and lines should not wrap

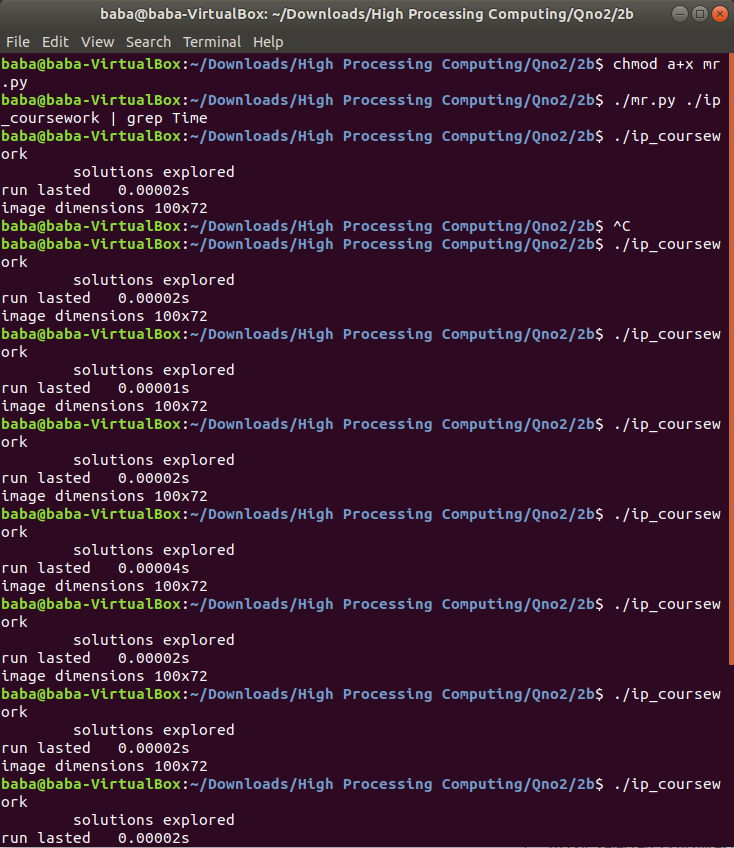
## 1.2 Image Processing

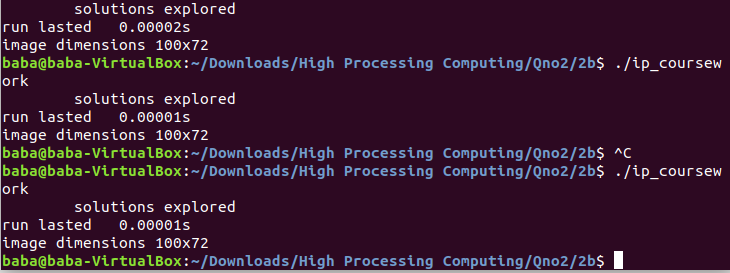
Insert the image displayed by your program

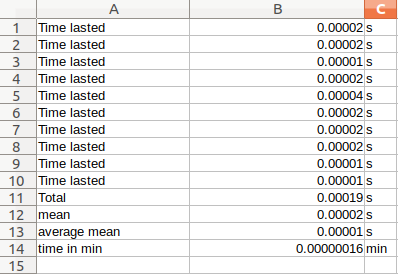


Insert a table that has columns containing running times for the original program and your multithread version. Mean running times should be included at the bottom of the columns.

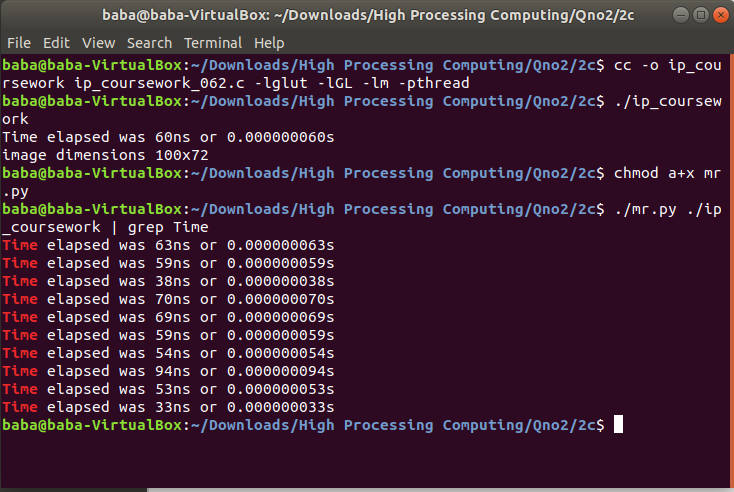
Original Version

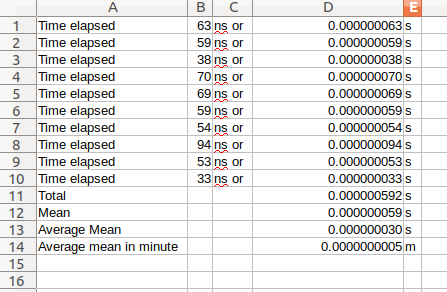






Multi-Threading Veersion





|  |
| --- |
| #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>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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 ip\_coursework\_062.c -lglut -lGL -lm -pthread    Dr Kevan Buckley, University of Wolverhampton, 2018  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define width 100  #define height 72  typedef struct arguments\_t{  int start;  int stride;  }arguments\_t;  unsigned char image[], results[width \* height];  void \*detect\_edges(unsigned char \*in, unsigned char \*out, arguments\_t \*args ) {  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 \*find\_factors(void \*args){  detect\_edges(image,results,args);  }  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) {  signal(SIGINT, sigint\_callback);  glutInit(&argc, argv);  struct timespec start, finish;  long long int time\_elapsed;  clock\_gettime(CLOCK\_MONOTONIC, &start);    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);      pthread\_t t1, t2, t3, t4;  arguments\_t t1\_arguments;  t1\_arguments.start = 0;  t1\_arguments.stride = 4;  arguments\_t t2\_arguments;  t2\_arguments.start = 1;  t2\_arguments.stride = 4;  arguments\_t t3\_arguments;  t3\_arguments.start = 2;  t3\_arguments.stride = 4;  arguments\_t t4\_arguments;  t4\_arguments.start = 3;  t4\_arguments.stride = 4;  void \*find\_factors();    pthread\_create(&t1, NULL, find\_factors, &t1\_arguments);  pthread\_create(&t2, NULL, find\_factors, &t2\_arguments);  pthread\_create(&t3, NULL, find\_factors, &t3\_arguments);  pthread\_create(&t4, NULL, find\_factors, &t4\_arguments);  pthread\_join(t1, NULL);  pthread\_join(t2, NULL);  pthread\_join(t3, NULL);  pthread\_join(t4, NULL);    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,  0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,  255,0,0,0,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,0,  255,255,255,255,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  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,  0,255,255,0,255,255,255,255,255,255,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  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,  0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  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,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,0,255,0,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,255,255,0,0,0,255,0,255,255,0,0,0,0,0,  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,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,255,0,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,  255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,  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,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  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,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,0,0,  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,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,255,255,255,255,255,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,  0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,  255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,  0,0,255,0,0,0,0,255,0,0,0,0,0,0,255,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,255,  255,255,255,0,255,255,0,0,0,0,0,255,255,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,255,255,0,255,255,255,255,0,0,255,0,0,0,0,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  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,  0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,0,0,  255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  0,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,255,0,0,  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,  0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  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,  0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,255,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,  0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,0,255,0,  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,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  255,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,255,255,0,255,0,0,0,0,0,0,  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,  0,0,255,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,255,255,255,255,0,255,0,0,0,0,0,0,0,0,0,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  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,  255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  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,  0,0,0,0,0,255,255,255,255,255,255,255,255,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  255,255,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,255,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,  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  }; |

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

## 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.

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.