

High Performance Computing

(6CS005)

Student Id : 1928930

Student Name : Bishal Khadka

Group : C3G1

Module leader: Mr. Jnaneshwar Bohara

Date of submission: 01/04/2020

Submitted on : <dd-mm-yy>

# Table of Contents

[Table of Contents 2](#_Toc29112109)

[1. POSIX Threads 1](#_Toc29112110)

[1.1 Password Cracking 1](#_Toc29112111)

[1.2 Image Processing 12](#_Toc29112112)

[1.3 Linear Regression 30](#_Toc29112113)

[2. CUDA 48](#_Toc29112114)

[2.1 Password Cracking 48](#_Toc29112115)

[2.2 Image Processing 57](#_Toc29112116)

[2.3 Linear Regression 76](#_Toc29112117)

[3. MPI 100](#_Toc29112118)

[3.1 Password Cracking 100](#_Toc29112119)

[3.2 Image Processing 115](#_Toc29112120)

[3.3 Linear Regression 137](#_Toc29112121)

**Table of figure:**

[Figure 1 Time elapsed for three-initials password cracker 6](#_Toc29123094)

[Figure 2 Image displayed by our program 12](#_Toc29123095)

[Figure 3 Scatter plot of our data 30](#_Toc29123096)

[Figure 4 Guess 1 31](#_Toc29123097)

[Figure 5 Guess 2 32](#_Toc29123098)

[Figure 6 Guess 3 33](#_Toc29123099)

[Figure 7 Solution overlaid for linear regression 34](#_Toc29123100)

[Figure 8 Time elapsed for six-digit two initial with four digits 114](#_Toc29123101)

[Table 1.1 Running time for password cracking 1](#_Toc29112428)

[Table 2 Mean running time for password cracker using multithread 7](#_Toc29112429)

[Table 3 Running time comparisons between original program and multithread version 14](#_Toc29112430)

[Table 4 Running time for multithread 47](#_Toc29112431)

[Table 5 Running time for CUDA Password cracking 55](#_Toc29112432)

[Table 6 Comparision between original and multithread password cracking 56](#_Toc29112433)

[Table 7 Comparison between original and CUDA version of image processing 75](#_Toc29112434)

[Table 8 Comparision between original and cuda verison of linear regression 100](#_Toc29112435)

[Table 9 Comparison between original and MPI version password cracking 108](#_Toc29112436)

[Table 10 StartIndex and End index for all rank 135](#_Toc29112437)

[Table 11 Comparison between original and MPI version of Image processing 137](#_Toc29112438)

[Table 12 comparison of running time between original program and MPI version of linear regression 154](#_Toc29112439)

# 1. POSIX Threads

## Password Cracking

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

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Running time in nano second | Running time in second | Mean Running Time |
| 1 | 516887633220ns | 516.887633220s |  |
| 2 | 506811146889ns | 506.811146889s |  |
| 3 | 508927593166ns | 508.927593166s |  |
| 4 | 507269538405ns | 507.269538405s |  |
| 5 | 507184282818ns | 507.184282818s | 509838232472.4ns or 509.8382324724s |
| 6 | 508754546574ns | 508.754546574s |  |
| 7 | 508737758971ns | 508.737758971s |  |
| 8 | 509791337001ns | 509.791337001s |  |
| 9 | 517265658095ns | 517.265658095s |  |
| 10 | 506752829585ns | 506.752829585s |  |

Table 1.1 Running time for password cracking

**1.1.2 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 total time taken by the above program as shown in table above is 509838232472.4 nano seconds (509.8382324724 seconds). The above program helped us to crack the password with 2 initial digit and 2 number like PQ12. Password cracker program tries to find out the each and every possible combination of password starting from AA00 to ZZ99. Note that only Upper-case letter are used in the 2-initial digit. The password is clean text without any punctuation marks like apostrophe, hyphen, exclamatory, ampersand, question marks etc. This type of technique is called as brute force attack in the world of hacking. Since the loop for the 2 number is constant for both the password cracker programs and one another digit should be increased in new program to reach the number of 3 initial like BIS92. The number of complexities increase by one digit which is one of the capital alphabets from English alphabets. We also know that there are 26 letters in English alphabet ranging from A to Z. So, in order to find the one digit of the password, loop must encounter 26 times and time estimation can be calculated by using the following formula.

Time estimation for 3 initial 2 number = Mean time for 2 initial 2 number × 26

= 509.8382324724 × 26 seconds

= 13255.7940443 seconds

**1.1.3 Three- Initial Code:**

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4. #include <crypt.h>
5. **#include <time.h>**
6. */\*\*\**
7. *Cracking Password with three initial and two numbers encrypted Password by using a simple*
8. *"brute force" algorithm.Works on passwords that consist only of 3 uppercase*
9. *letters and a 2 digit integer.*
10. ***Compile with:***
11. *cc -o threeinitial threeinitial.c -lcrypt*
12. *To run and store the result in txt file:*
13. *./threeinitial > resultthreeinitial.txt*
14. *\*\*\*/*
15. **int no\_of\_passwords = 4;**
16. char \*encrypted\_passwords[] = {
17. "$6$KB$/B53sd.H45Cys4TU2/BQm.PcsoxGwNJWJfncz502dUks7KzswAQRhrtuMr2G1L17uPV05TkxzGLaanzmWf5.z0",
18. "$6$KB$nQJZ3xbUH4nWxYKkBtO1jIPjCkd6gZ00FKK88k2AFM43PGzxRHr34ZFNQcPMRp8pMiYDwiGlr2xtZg0heqv.I.",
19. "$6$KB$PLEw.VlzhUdt7y1Lo04CUpFjlZpLg3uXR4Ob77pw1YCZFYxf4rRwqjwW53gONjZYJpfTogYiXCQXe1mV2yito/",
20. **"$6$KB$9Kf9hYiattXoyh4rTJI2ORr9EXy.eK4kToz8xG8U83Ky6vAl3XwErM.393EgpdJ1mBssJ5u/pBWnlDdDKrma7."**
21. };
22. *//returns the substring of given string between two given indices.*
23. void substr(char \*dest, char \*src, int start, int length){
24. memcpy(dest, src + start, length);
25. **\*(dest + length) = '\0';**
26. }
27. */\*\**
28. *This function can crack the kind of password explained above. All combinations*
29. *that are tried are displayed and when the password is found, password found is put at the*
30. ***start of the line.***
31. *\*/*
32. void crack(char \*salt\_and\_encrypted){
33. int l1, l2, l3,l4; *// Loop counters*
34. char salt[7]; *// String used in hashing the password. Need space for \0*
35. **char plain[7]; *// The combination of letters currently being checked***
36. char \*enc; *// Pointer to the encrypted password*
37. int count = 0; *// The number of combinations explored so far*
39. substr(salt, salt\_and\_encrypted, 0, 6);
40. ***// l1, l2 and l3 are three loop variable***
41. for(l1='A'; l1<='Z'; l1++){
42. for(l2='A'; l2<='Z'; l2++){
43. for(l3='A'; l3<='Z'; l3++){
44. for(l4=0; l4<=99; l4++){
45. **sprintf(plain, "%c%c%c%02d", l1, l2, l3, l4);**
46. enc = (char \*) crypt(plain, salt);
47. count++;
48. if(strcmp(salt\_and\_encrypted, enc) == 0){
49. printf("Password Found%-8d%s %s**\n**", count, plain, enc);
50. **} else {**
51. printf(" %-8d%s %s**\n**", count, plain, enc);
52. }
53. }
54. }
55. **}**
56. }
57. printf("%d solutions explored**\n**", count);
58. }
60. **int time\_difference(struct timespec \*start, struct timespec \*finish,**
61. long long int \*difference) {
62. long long int ds = finish->tv\_sec - start->tv\_sec;
63. long long int dn = finish->tv\_nsec - start->tv\_nsec;
65. **if(dn < 0 ) {**
66. ds--;
67. dn += 1000000000;
68. }
69. \*difference = ds \* 1000000000 + dn;
70. **return !(\*difference > 0);**
71. }
73. int main(int argc, char \*argv[]){
74. *//calculating time difference*
75. **struct timespec start, finish;**
76. long long int time\_elapsed;
78. clock\_gettime(CLOCK\_MONOTONIC, &start);
79. int i;
80. **for(i=0;i<no\_of\_passwords;i<i++) {**
81. crack(encrypted\_passwords[i]);
82. }
83. clock\_gettime(CLOCK\_MONOTONIC, &finish);
84. time\_difference(&start, &finish, &time\_elapsed);
85. **printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,**
86. (time\_elapsed/1.0e9));
87. return 0;
88. }

**1.1.4 Explanation of Three- Initial Code:**

There are many ways of cracking password in real world. Since we are trying to crack passwords that are very short in length, Brute force attack method is used. The password we are trying to crack contains only three capital letter initials followed by 2 numbers. Loop will be used to find all the possible passwords with that combination until correct one is found. We are trying to crack 4 passwords and calculate the total time to find all the combination of passwords. Steps involved in password cracking are:

1. Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. CLOCK\_MONOTIC is used to get the absolute time right after the execution of program. CLOCK\_MONOTONIC is used instead of CLOCK\_REALTIME because CLOCK\_MONOTONIC is not affected by the system data and time. Time elapsed is calculated by subtracting finish time and start time after getting the both the time.

2. The variable i has been declared with data type int to crack four passwords in the loop. The loop will run from i=0 to i=3 since we have declared the value of n\_password to 4. During all four-iteration crack function will be executed. The crack function is user-defined function and array of encrypted password is passed to the non-value returning(void) crack function. Inside crack function, four loops are declared to find the correct password. The first, second and third loops are for the first alphabet, second alphabet and third alphabet ranging from A-Z and the fourth loop is for the 2 digits ranging from 00-99. The nested for loop is used to get all the combination of passwords.

3. We have used encryption along with six-digit ($6$KB$) salt in our password. So, in order to compare our loop encountered password with the provided password we must add salt with same six-digit. substr() function is used to separate first six digit from the salt and encrypted password and memcpy() function inside the substr() function with parameter destination, source and number is used to copy only the salt strings used in the encrypted password. And another function crypt()is used to encrypt the salt added plain password which later is compared to our password to find out the matching password. \*enc is the pointer to the encrypted password and count is the number of times the combination has explored. Strcmp() is another library function of c language which is used to compare the value of two variable. Here, we have used two variable salt\_and\_encrypted is the first string which has to be compared to another string named enc. In Strcmp() library function, it returns value zero if both strings are equal. So, if only the same value found, password found string is put before the count variable else other value like count, plain password and encrypted password is printed. Again, same process is repeated for four times to find each password in each loop. After completion of four loop, the total time taken by the program is calculated by the function named time\_difference() function.

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

My assumption for new password cracker program would be 26 times more time than the old original program which is 13255.7940443 seconds. After running the code of three-initial, my earlier hypothesis supports my assumption. The running time for three initials is found to be 13185.474021264 seconds which is almost equal to the timing of assumption. The difference of run time is only 70.320023036 ≈ 70 seconds ≈ 1minute. Therefore my assumption corroborates the actual run time of three initial.

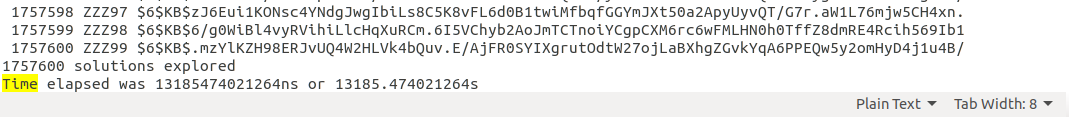


Figure 1 Time elapsed for three-initials password cracker

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

The runtime for original password is in table 1.1 and mean run time for original program is found to be 509838232472.4ns or 509.8382324724s and the runtime for password cracker is shown in below table 2.

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Running time in nano second | Running time in second | Mean Running Time |
| 1 | 255709941192ns | 255.709941192s |  |
| 2 | 271617276815ns | 271.617276815s |  |
| 3 | 252963400124ns | 252.963400124s |  |
| 4 | 253174073696ns | 253.174073696s |  |
| 5 | 262301072818ns | 262.301072818s | 258555707343.5 ns or 258.5557073435 s |
| 6 | 265985488125ns | 265.985488125s |  |
| 7 | 260231187042ns | 260.231187042s |  |
| 8 | 253731283786ns | 253.731283786s |  |
| 9 | 253239442836ns | 253.239442836s |  |
| 10 | 256603907001ns | 256.603907001s |  |

Table 2 Mean running time for password cracker using multithread

From above table 2 it is clearly seen that the mean running time for password cracker using multithread is 258555707343.5 nanosecond or 258.5557073435 second which is (509.8382324724s - 258.5557073435s) = 251.282525129s. In second program we used two threads in a single program to find the matching password that can run simultaneously. One thread runs the first loop from ‘A’ to ‘M’ and another thread runs the first loop from ‘N’ to ‘Z’ at the same time so, the time taken by second program is almost half the time taken by the first program.

**1.1.6 Code For multithread password cracker:**

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4. **#include <crypt.h>**
5. #include <time.h>
6. #include <pthread.h>
7. */\*\*\**
8. *Cracking Password using thread with two initial and two numbers encrypted Password by using a simple*
9. ***"brute force" algorithm .Works on passwords that consist only of 2 uppercase***
10. *letters and a 2 digit integer.*
11. *Compile with:*
12. *cc -o thread thread.c -lcrypt -pthread*
13. *To run and store the result in txt file:*
14. ***./thread > p1.txt***
15. *\*\*\*/*
17. int no\_of\_passwords = 4;
19. **char \*encrypted\_passwords[] = {**
20. "$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",
21. "$6$KB$VDUCASt5S88l82JzexhKDQLeUJ5zfxr16VhlVwNOs0YLiLYDciLDmN3QYAE80UIzfryYmpR.NFmbZvAGNoaHW.",
22. "$6$KB$LIQ21WOW2T678MQ//wddYygf4uuWaMWVM8vG9Pj9O.7sDQ1GVgmw8KqqNfjjU6BtBgJvE5nXbborTU/uwrnVz1",
23. "$6$KB$cU/6LrcqQGvsvRGllyfGDU1Wuvd1roXp.IEr3UYKQjBbT1sdP1PZPrZKrTGfFuRqD97EsgDalOwwn3OedIIjn0"
24. **};**
26. */\*\**
27. *Required by lack of standard function in C.*
28. *\*/*
30. void substr(char \*dest, char \*src, int start, int length){
31. memcpy(dest, src + start, length);
32. \*(dest + length) = '**\0**';
33. }
35. */\*\**
36. *This function can crack the kind of password explained above. All combinations*
37. *that are tried are displayed and when the password is found Password found, is put at the*
38. *start of the line.*
39. ***\*/***
41. void thread(char \*encrypted\_passwords){
43. pthread\_t Th1,Th2;
45. void \*kernel\_function\_1();
46. void \*kernel\_function\_2();
48. pthread\_create(&Th1, NULL, kernel\_function\_1, (void \*)encrypted\_passwords);
49. **pthread\_create(&Th2, NULL, kernel\_function\_2, (void \*)encrypted\_passwords);**
51. pthread\_join(Th1, NULL);
52. pthread\_join(Th2, NULL);
53. }
55. void \*kernel\_function\_1(char \*salt\_and\_encrypted){
56. int bi, sh, al; *// Loop counters*
57. char salt[7]; *// String used in hashing the password. Need space for \0*
58. char plain[7]; *// The combination of letters currently being checked*
59. **char \*enc; *// Pointer to the encrypted password***
60. int count = 0; *// The number of combinations explored so far*
62. substr(salt, salt\_and\_encrypted, 0, 6);
64. **for(bi='A'; bi<='M'; bi++){**
65. for(sh='A'; sh<='Z'; sh++){
66. for(al=0; al<=99; al++){
67. sprintf(plain, "%c%c%02d", bi, sh, al);
68. enc = (char \*) crypt(plain, salt);
69. **count++;**
70. if(strcmp(salt\_and\_encrypted, enc) == 0){
71. printf("Password found %-8d%s %s**\n**", count, plain, enc);
72. } else {
73. printf(" %-8d%s %s**\n**", count, plain, enc);
74. **}**
75. }
76. }
77. }
78. printf("%d solutions explored**\n**", count);
79. **}**
81. void \*kernel\_function\_2(char \*salt\_and\_encrypted){
82. int bi, sh, al; *// Loop counters*
83. char salt[7]; *// String used in hashing the password. Need space for \0*
84. **char plain[7]; *// The combination of letters currently being checked***
85. char \*enc; *// Pointer to the encrypted password*
86. int count = 0; *// The number of combinations explored so far*
88. substr(salt, salt\_and\_encrypted, 0, 6);
90. for(bi='N'; bi<='Z'; bi++){
91. for(sh='A'; sh<='Z'; sh++){
92. for(al=0; al<=99; al++){
93. sprintf(plain, "%c%c%02d", bi, sh, al);
94. **enc = (char \*) crypt(plain, salt);**
95. count++;
96. if(strcmp(salt\_and\_encrypted, enc) == 0){
97. printf("Password found %-8d%s %s**\n**", count, plain, enc);
98. } else {
99. **printf(" %-8d%s %s\n", count, plain, enc);**
100. }
101. }
102. }
103. }
104. **printf("%d solutions explored\n", count);**
105. }
106. int time\_difference(struct timespec \*start,
107. **struct timespec \*finish,**
108. long long int \*difference) {
109. long long int ds = finish->tv\_sec - start->tv\_sec;
110. long long int dn = finish->tv\_nsec - start->tv\_nsec;
112. **if(dn < 0 ) {**
113. ds--;
114. dn += 1000000000;
115. }
116. \*difference = ds \* 1000000000 + dn;
117. **return !(\*difference > 0);**
118. }
119. int main(int argc, char \*argv[]){
120. int i;
121. *//calculating time difference*
122. **struct timespec start, finish;**
123. long long int time\_elapsed;
125. clock\_gettime(CLOCK\_MONOTONIC, &start);
126. for(i=0;i<n\_passwords;i<i++) {
127. **thread(encrypted\_passwords[i]);**
128. }
129. clock\_gettime(CLOCK\_MONOTONIC, &finish);
130. time\_difference(&start, &finish, &time\_elapsed);
131. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed,
132. **(time\_elapsed/1.0e9));**
133. return 0;
134. }

**1.1.7 Explanation For multithread password cracker:**

Multithread password cracker works similar to the original two initials password. The only difference is we used multi thread to minimize the runtime and to boost up the performance of program because instead of running one thread multiple thread can be run simultaneously in a single program. Just like normal program, time is calculated same here. Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. CLOCK\_MONOTIC is used to get the absolute time right after the execution of program. CLOCK\_MONOTONIC is used instead of CLOCK\_REALTIME because CLOCK\_MONOTONIC is not affected by the system data and time. Time elapsed is calculated by subtracting finish time and start time after getting the both the time. Encrypted password with salt is sent to the thread function from the for loop. Inside thread function, pthread\_t datatype is assigned to two threads Th1 and Th2. Before using threads in c language, #include <pthread.h> must be included in the header section of the program. Two kernel functions named kernel\_function\_1() and kernel\_function\_2() with void return type. Right after that we have created the thread with pthread\_create() function which takes four parameters. First parameter is identifier for new thread, second parameter is thread attribute object, third parameter is start routine and fourth parameter is single argument passed to third parameter that is start routine. In our case we have Th1, NULL, kernel\_function\_1 and encrypted\_passwords are four parameters respectively for thread 1. Similarly, for second thread the parameters are Th2, NULL, kernel\_function\_2 and encrypted\_passwords respectively. Then kernel\_function\_1() and kernel\_function\_2() method has two different for loop. First kernel function starts the first loop form ‘A’ to ‘M’ and second kernel function starts the first loop from’ N’ to ‘Z’ to match the given passwords. The procedure to find out the match is same as described in original program above.

We have used encryption along with six-digit ($6$KB$) salt in our password. So, in order to compare our loop encountered password with the provided password we must add salt with same six-digit. substr() function is used to separate first six digit from the salt and encrypted password and memcpy() function inside the substr() function with parameter destination, source and number is used to copy only the salt strings used in the encrypted password. And another function crypt()is used to encrypt the salt added plain password which later is compared to our password to find out the matching password. \*enc is the pointer to the encrypted password and count is the number of times the combination has explored. Strcmp() is another library function of c language which is used to compare the value of two variable. Here, we have used two variable salt\_and\_encrypted is the first string which has to be compared to another string named enc. In Strcmp() library function, it returns value zero if both strings are equal. So, if only the same value found, password found string is put before the count variable else other value like count, plain password and encrypted password is printed. Again, same process is repeated for four times to find each password in each loop. After completion of four loop, the total time taken by the program is calculated by the function named time\_difference() function.

Since we have two threads, pthread\_join is used to join two threads. pthread\_join is library function which takes thread and value\_ptr as argument. In our case Th1 and Th2 are two arguments for the thread and value\_ptr is set to NULL. This same process is repeated for four times to find each password in each loop. After completion of four loop, the total time taken by the program is calculated by the function named time\_difference() function.

## Image Processing

**Insert the image displayed by your program**

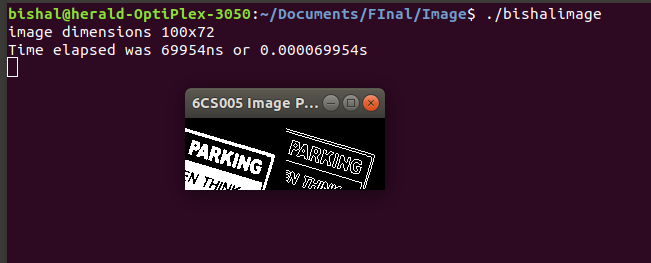


Figure 2 Image displayed by our program

**Code of multithread image processing:**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <GL/glut.h>
5. **#include <GL/gl.h>**
6. #include <malloc.h>
7. #include <signal.h>
9. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
10. ***Displays two grey scale images. On the left is an image that has come from an***
11. *image processing pipeline, just after colour thresholding. On the right is*
12. *the result of applying an edge detection convolution operator to the left*
13. *image. This program performs that convolution.*

16. *To compile adapt the code below wo match your filenames:*
17. *cc -o bishalimage bishal046.c -lglut -lGL -lm*
19. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
20. **#define width 100**
21. #define height 72
23. unsigned char image[], results[width \* height];
25. **void detect\_edges(unsigned char \*in, unsigned char \*out) {**
26. int i;
27. int n\_pixels = width \* height;
29. for(i=0;i<n\_pixels;i++) {
30. **int x, y; // the pixel of interest**
31. int b, d, f, h; // the pixels adjacent to x,y used for the calculation
32. int r; // the result of calculate
34. y = i / width;
35. **x = i - (width \* y);**
37. if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {
38. results[i] = 0;
39. } else {
40. **b = i + width;**
41. d = i - 1;
42. f = i + 1;
43. h = i - width;
45. **r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)**
46. + (in[h] \* -1);
48. if (r > 0) { // if the result is positive this is an edge pixel
49. out[i] = 255;
50. **} else {**
51. out[i] = 0;
52. }
53. }
54. }
55. **}**
57. void tidy\_and\_exit() {
58. exit(0);
59. }
61. void sigint\_callback(int signal\_number){
62. printf("**\n**Interrupt from keyboard**\n**");
63. tidy\_and\_exit();
64. }
66. static void display() {
67. glClear(GL\_COLOR\_BUFFER\_BIT);
68. glRasterPos4i(-1, -1, 0, 1);
69. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);
70. **glRasterPos4i(0, -1, 0, 1);**
71. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);
72. glFlush();
73. }
75. **static void key\_pressed(unsigned char key, int x, int y) {**
76. switch(key){
77. case 27: // ASCII value for escape
78. tidy\_and\_exit();
79. break;
80. **default:**
81. printf("**\n**Press escape to exit**\n**");
82. break;
83. }
84. }
85. **int time\_difference(struct timespec \*start, struct timespec \*finish,**
86. long long int \*difference) {
87. long long int ds = finish->tv\_sec - start->tv\_sec;
88. long long int dn = finish->tv\_nsec - start->tv\_nsec;
90. **if(dn < 0 ) {**
91. ds--;
92. dn += 1000000000;
93. }
94. \*difference = ds \* 1000000000 + dn;
95. **return !(\*difference > 0);**
96. }
97. int main(int argc, char \*\*argv) {
98. **signal(SIGINT, sigint\_callback);**
99. //calculating time
100. struct timespec start, finish;
101. long long int time\_elapsed;
103. **clock\_gettime(CLOCK\_MONOTONIC, &start);**
104. printf("image dimensions %dx%d**\n**", width, height);
105. detect\_edges(image, results);
106. clock\_gettime(CLOCK\_MONOTONIC, &finish);
107. time\_difference(&start, &finish, &time\_elapsed);
108. **printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,**
109. (time\_elapsed/1.0e9));
110. glutInit(&argc, argv);
111. glutInitWindowSize(width \* 2,height);
112. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);
114. glutCreateWindow("6CS005 Image Progessing Bishal");
115. glutDisplayFunc(display);
116. glutKeyboardFunc(key\_pressed);
117. glClearColor(0.0, 1.0, 0.0, 1.0);
119. glutMainLoop();
121. tidy\_and\_exit();
123. **return 0;**
124. }
125. unsigned char image[] = {255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
126. 255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,255,255,
127. 0,255,255,255,0,0,0,0,255,255,0,0,0,0,0,0,255,255,255,
128. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
129. 255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
130. 0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,
131. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,
132. 255,0,0,255,255,0,0,255,255,0,255,255,0,0,255,0,0,255,255,
133. 0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
134. 255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,
135. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,
136. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,
137. 255,255,255,0,0,255,255,0,0,255,255,0,255,0,0,255,255,0,0,
138. 255,255,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,
139. 255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,
140. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
141. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,
142. 255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,0,0,0,
143. 0,255,255,255,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,
144. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,
145. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
146. 0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
147. 255,255,0,255,255,255,255,255,0,0,255,255,255,0,0,0,255,255,0,
148. 0,255,0,0,0,0,255,255,255,0,0,255,255,0,255,255,255,255,255,
149. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
150. 255,255,255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,
151. 0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,
152. 255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,
153. 0,0,0,255,0,0,255,255,0,0,0,255,255,255,255,0,255,255,255,
154. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
155. 255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,
156. 0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,
157. 255,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,
158. 0,0,0,255,255,255,0,0,255,255,0,0,255,0,0,0,255,255,255,
159. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
160. 255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,
161. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,
162. 255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,0,
163. 0,255,255,255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,
164. 0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
165. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,
166. 0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
167. 0,0,0,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,255,
168. 255,255,255,255,255,0,255,255,255,255,255,0,255,255,255,255,255,0,0,
169. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
170. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
171. 255,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,
172. 0,0,0,0,0,0,0,0,0,255,255,0,0,255,255,255,0,0,0,
173. 255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,255,
174. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
175. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
176. 255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,
177. 0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,
178. 255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,
179. 255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
180. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
181. 255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,
182. 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,
183. 255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,
184. 255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,
185. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
186. 255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
187. 0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
188. 0,0,0,0,255,255,255,255,255,0,255,0,0,255,255,0,0,255,255,
189. 255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,
190. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
191. 255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,
192. 0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,
193. 0,0,0,0,0,0,0,0,0,255,0,0,255,255,0,0,0,0,255,
194. 255,255,0,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,
195. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
196. 255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,
197. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,
198. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,
199. 255,0,0,0,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,
200. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
201. 255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,
202. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
203. 255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
204. 255,255,255,255,255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,
205. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
206. 255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
207. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
208. 0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,
209. 0,0,0,0,0,255,255,255,255,255,255,0,0,0,255,255,255,255,255,
210. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
211. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,
212. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
213. 255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,
214. 0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,255,255,0,255,
215. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
216. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,
217. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
218. 255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,
219. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
220. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
221. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,
222. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
223. 0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,
224. 0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
225. 0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
226. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
227. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
228. 0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,255,255,255,
229. 255,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,
230. 0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,
231. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
232. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
233. 0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,255,255,0,
234. 0,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,0,0,0,
235. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,
236. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
237. 255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
238. 0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,0,
239. 255,255,255,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,255,
240. 255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
241. 255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
242. 255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
243. 0,0,0,0,0,0,0,255,255,0,0,255,255,255,255,0,0,255,255,
244. 255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,255,255,0,0,
245. 0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,
246. 0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
247. 255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,
248. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,
249. 255,0,0,255,255,255,255,255,255,0,255,255,255,255,0,0,255,255,255,
250. 255,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,
251. 0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,
252. 255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,
253. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,
254. 255,0,0,255,255,255,0,0,255,255,255,255,255,255,0,255,255,255,255,
255. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,
256. 0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,
257. 255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,
258. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
259. 255,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,
260. 255,255,255,255,255,0,0,0,0,0,255,255,255,0,0,0,0,0,0,
261. 0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,
262. 255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,
263. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,
264. 0,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,
265. 255,255,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,
266. 0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,
267. 0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
268. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,
269. 255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,0,
270. 255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,
271. 255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,
272. 0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,
273. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
274. 255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,0,0,0,255,
275. 255,255,255,0,0,255,255,255,255,255,255,255,0,255,255,255,0,0,255,
276. 255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,255,
277. 255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,
278. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
279. 255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,0,255,255,255,
280. 255,0,0,0,0,255,255,255,255,0,255,255,255,255,255,255,0,0,255,
281. 255,255,255,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
282. 0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,
283. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
284. 0,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,
285. 255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,
286. 255,255,0,0,255,255,255,255,0,0,0,255,255,255,255,255,0,0,0,
287. 0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,
288. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
289. 0,0,0,0,255,255,255,255,0,255,255,255,0,0,0,255,255,255,0,
290. 0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,
291. 0,0,255,255,255,255,255,0,0,0,255,255,255,0,0,0,0,0,0,
292. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,
293. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
294. 0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,
295. 0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,
296. 0,0,255,255,255,0,0,255,255,255,255,255,0,0,0,255,255,255,0,
297. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
298. 255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
299. 0,0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,
300. 255,255,255,255,0,255,255,255,255,0,0,0,0,255,255,255,255,255,255,
301. 255,255,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,
302. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
303. 0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,
304. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,
305. 255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,0,0,0,255,
306. 255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,0,255,
307. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
308. 0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,
309. 0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,255,255,
310. 255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,
311. 255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,
312. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
313. 0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,
314. 0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,
315. 0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,
316. 255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,255,255,
317. 255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
318. 0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,
319. 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
320. 255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,
321. 0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,
322. 0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,
323. 0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,
324. 255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
325. 0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,
326. 0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,
327. 0,255,255,255,255,0,0,0,255,255,0,0,0,0,0,0,0,0,0,
328. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,
329. 255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
330. 0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,0,
331. 0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,
332. 255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,
333. 0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,
334. 255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
335. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,
336. 255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,0,0,0,255,
337. 255,255,255,255,255,255,255,255,255,0,0,0,255,0,0,0,0,0,0,
338. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,
339. 255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,
340. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
341. 0,0,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,
342. 255,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
343. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
344. 255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,
345. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
346. 0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,255,
347. 255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,0,0,0,
348. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,
349. 255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,
350. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
351. 0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,255,
352. 255,255,255,0,0,255,255,255,255,255,255,0,0,0,0,255,255,255,0,
353. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
354. 255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
355. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
356. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,
357. 255,255,0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,0,
358. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
359. 0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,
360. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
361. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
362. 0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,
363. 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
364. 0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,
365. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
366. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
367. 0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,
368. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
369. 0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,
370. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
371. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
372. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,
373. 255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
374. 0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,
375. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
376. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
377. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
378. 0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,
379. 0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,
380. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
381. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
382. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
383. 0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,
384. 0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,
385. 255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
386. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
387. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
388. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
389. 0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,
390. 255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
391. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
392. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
393. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
394. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,
395. 255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,
396. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
397. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
398. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
399. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,
400. 255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,
401. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
402. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
403. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
404. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,
405. 255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,
406. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
407. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
408. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
409. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,
410. 255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,
411. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
412. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
413. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
414. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
415. 0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,
416. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
417. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
418. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
419. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
420. 0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,
421. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
422. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
423. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
424. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
425. 0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,
426. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
427. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
428. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
429. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
430. 0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,
431. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
432. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
433. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
434. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
435. 0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,
436. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
437. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
438. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
439. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
440. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,
441. 255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
442. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
443. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
444. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
445. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
446. 0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
447. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
448. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
449. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
450. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
451. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
452. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
453. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
454. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
455. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
456. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
457. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
458. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
459. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
460. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
461. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
462. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
463. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
464. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
465. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
466. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
467. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
468. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
469. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
470. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
471. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
472. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
473. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
474. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
475. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
476. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
477. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
478. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
479. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
480. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
481. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
482. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
483. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
484. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
485. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
486. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
487. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
488. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
489. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
490. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
491. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
492. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
493. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
494. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
495. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
496. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
497. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
498. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
499. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
500. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
501. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
502. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
503. 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
504. };

**Explanation of code:**

The main idea behind the code is to know how computer stores and manipulate two-dimensional array of cells called image or picture. Picture is made up of pixels which is made up of horizontal and vertical coordinate x and y respectively.

Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. CLOCK\_MONOTIC is used to get the absolute time right after the execution of program. CLOCK\_MONOTONIC is used instead of CLOCK\_REALTIME because CLOCK\_MONOTONIC is not affected by the system data and time. Time elapsed is calculated by subtracting finish time and start time after getting the both the time.

Four threads are declared named T1, T2, T3, T4 with pthread\_t datatype. Before using threads in c language, #include <pthread.h> must be included in the header section of the program. The thread was created with pthread\_create() function which takes four parameters. First parameter is identifier for new thread, second parameter is thread attribute object, third parameter is start routine and fourth parameter is single argument passed to third parameter that is start routine. In our case we have

1. &T1, NULL, find, &T1\_arguments for first thread

2. &T2, NULL, find, &T2\_arguments for second thread

3. &T3, NULL, find, &T2\_arguments for third thread

4. &T4, NULL, find, &T2\_arguments for fourth thread

Find() function is defined with void return type. Inside find function detect\_edges function is defined which requires the three-parameter image, result and argument. Image is denoted by in pointer, result is denoted by out pointer and argument is denoted by arg pointer in detect-edges function. First of all, the number of pixels is calculated by multiplying width and height of picture which is set to 100 and 72 respectively at the top of the program. Our program uses convolution formula to detect the edge of the given picture. To detect the edge x, y pixels are taken from the image and defined as the

y = i / width;

x = i - (width \* y);

The value of result can be calculated by using the value obtaining formula given by convolution theorem:

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

If the value of r is greater than 0 that is positive, edge of output result is set to white color(255) and if value is not equal to zero then the output result will be set to black color. Since we have four threads, pthread\_join is used to join four threads to get the desired output image. The image is then showed to the windows with the help of utility kit GLUT. Different function of GLUT like glutInit, glutInitWindowSize, glutInitDisplayMode, glutCreateWindow, glutDisplayFunc, glutKeyboardFunc, glClearColor, glutMainLoop etc. are used to display the image.

**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.**

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (multithread) |
| 1 | 65933ns or 0.000065933s | 638556ns or 0. 000638556s |
| 2 | 115084ns or 0.000115084s | 4340169ns or 0. 004340169s |
| 3 | 64197ns or 0.000064197s | 2492724ns or 0. 002492724s |
| 4 | 65573ns or 0.000065573s | 485682ns or 0. 000485682s |
| 5 | 66473ns or 0.000066473s | 3574456ns or 0. 003574456s |
| 6 | 122975ns or 0.000122975s | 5617945ns or 0. 005617945s |
| 7 | 94871ns or 0.000094871s | 5606458ns or 0. 005606458s |
| 8 | 68225ns or 0.000068225s | 3571468ns or 0. 003571468s |
| 9 | 63991ns or 0.000063991s | 2247745ns or 0. 002247745s |
| 10 | 63396ns or 0.000063396s | 1128084ns or 0.001128084s |
|  | Mean Running time for original program: 79071.8ns or 0.0000790718s | Mean Running time for multithread program: 2970328.7ns or 0.0029703287s |

Table 3 Running time comparisons between original program and multithread version

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

After seeing the result, the mean running time multithread program took 2970328.7ns or 0.0029703287s whereas the mean Running time for original program is only 79071.8ns or 0.0000790718s. The multithread program for image processing took more time than the original program which seems to be abnormal in our case. The time taken by the original program is one thousand fractions faster than the multithread program. This may be due to the use of four thread for the small image with width only 100. The multithread program divides 100 to 4 parts that is 25 pixels for each 4 threads and tries to solve parallelly. It would take more time comparative to 100 pixels. Moreover, the multithread program takes couple of second to create and join multiple threads.

## Linear Regression

**Insert a scatter plot of your data.**

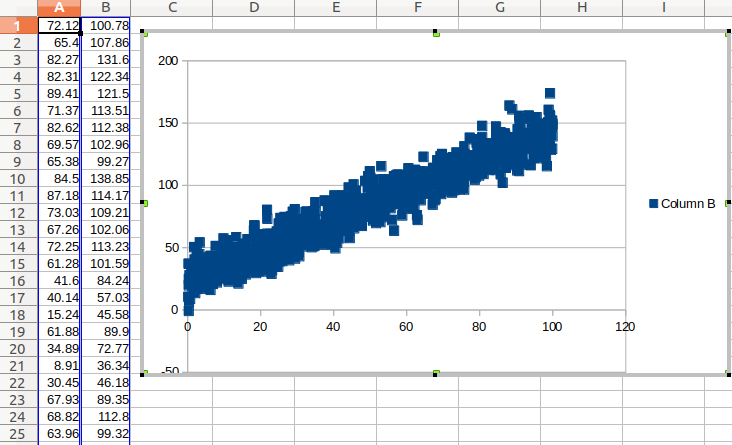


Figure 3 Scatter plot of our data

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

Three guesses at the optimum values for m and c are:  
**1. First Guess:**

m=1.25 c=0

**2. Second guess:**

m=1.25 c=25

**3. Third Guess:**

m= 1.25 c=50

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

**Guess 1:**

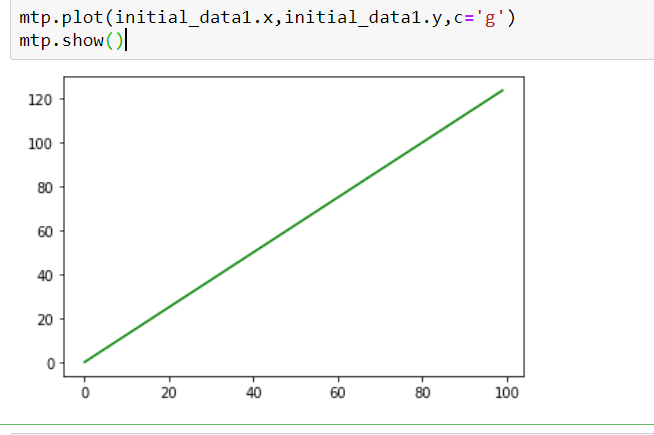


Figure 4 Guess 1

**Guess 2:**

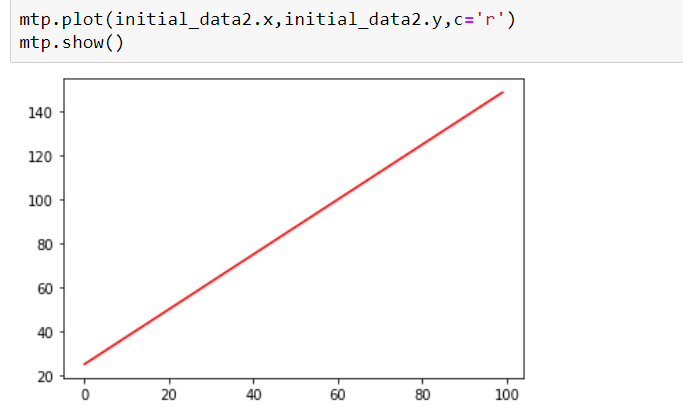


Figure 5 Guess 2

**Guess 3:**

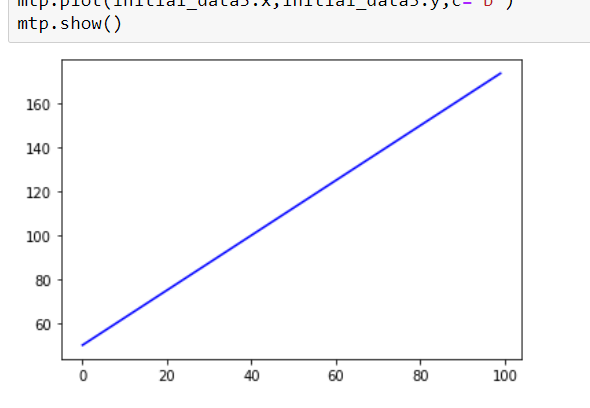
****

Figure 6 Guess 3

**Insert a graph that represets your data with the solution overlaid.**

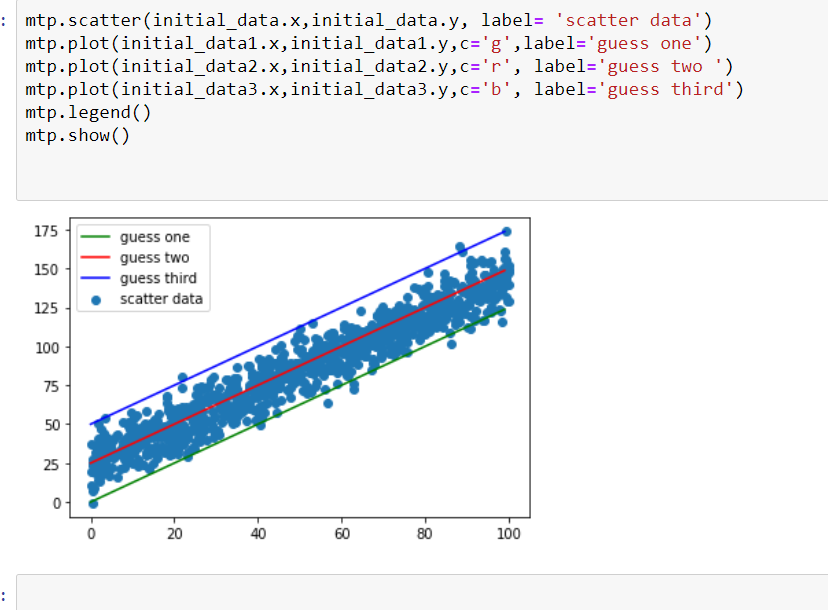


Figure 7 Solution overlaid for linear regression

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

I have guessed the line by guessing the value of slope and intercept. Intercept is the point where the x-axis and y-axis meet. Slope is the line that defines the steepness and direction of straight line. Slope can be calculated by using the formula:

Slope(m) = . Here, we have

y2= 175 y1= 50

x2= 100 x1 = 0

we get

. = 1.25.

Intercept is the point where x and y- axis meet. So, we tried this by using hit and trial method. In first guess we used the lowest value of the dot that is 0 and in second guess we chose the middle value that is 25 and in third guess we chose the highest value that is 50. So, the result after guessing seems to be correct.

**Code for multithread linear regression:**

1. #include <stdio.h>
2. #include <math.h>
3. #include <time.h>
4. #include <pthread.h>
6. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
7. *\* This program takes an initial estimate of m and c and finds the associated*
8. *\* rms error. It is then as a base to generate and evaluate 8 new estimates,*
9. *\* which are steps in different directions in m-c space. The best estimate is*
10. ***\* then used as the base for another iteration of "generate and evaluate". This***
11. *\* continues until none of the new estimates are better than the base. This is*
12. *\* a gradient search for a minimum in mc-space.*
13. *\**
14. *\* To compile:*
15. ***\* cc -o bishalthread bishalthread.c -lm -pthread***
16. *\**
17. *\* To run:*
18. *\* ./bishalthread*
19. *\**
20. ***\* Dr Kevan Buckley, University of Wolverhampton, 2018***
21. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
22. int i;
23. double bm = 1.3;
24. double bc = 10;
25. **double be;**
26. double dm[8];
27. double dc[8];
28. double e[8];
29. double step = 0.01;
30. **double best\_error = 999999999;**
31. int best\_error\_i;
32. int minimum\_found = 0;
33. double om[] = {0,1,1, 1, 0,-1,-1,-1};
34. double oc[] = {1,1,0,-1,-1,-1, 0, 1};
36. typedef struct point\_t {
37. double x;
38. double y;
39. } point\_t;
41. int n\_data = 1000;
42. point\_t data[];
44. double residual\_error(double x, double y, double m, double c) {
45. **double e = (m \* x) + c - y;**
46. return e \* e;
47. }
49. double rms\_error(double m, double c) {
50. **int i;**
51. double mean;
52. double error\_sum = 0;
54. for(i=0; i<n\_data; i++) {
55. **error\_sum += residual\_error(data[i].x, data[i].y, m, c);**
56. }
58. mean = error\_sum / n\_data;
60. **return sqrt(mean);**
61. }
62. int time\_difference(struct timespec \*start, struct timespec \*finish, long long int \*difference)
63. {
64. long long int ds = finish->tv\_sec - start->tv\_sec;
65. **long long int dn = finish->tv\_nsec - start->tv\_nsec;**
67. if(dn < 0 ) {
68. ds--;
69. dn += 1000000000;
70. **}**
71. \*difference = ds \* 1000000000 + dn;
72. return !(\*difference > 0);
73. }
75. **void \*linear\_regression\_thread(void \*args){**
77. int \*a = args;
78. int i = \*a;
80. **dm[i] = bm +(om[i] \* step);**
81. dc[i] = bc + (oc[i] \* step);
82. e[i] = rms\_error(dm[i], dc[i]);
83. if(e[i] < best\_error) {
84. best\_error = e[i];
85. **best\_error\_i = i;**
86. pthread\_exit(NULL);
87. }
88. }
90. **int main() {**
91. struct timespec start, finish;
92. long long int time\_elapsed;
93. clock\_gettime(CLOCK\_MONOTONIC, &start);
95. **int i;**
96. pthread\_t p\_threads[0];
98. be = rms\_error(bm, bc);
100. **while(!minimum\_found) {**
101. for(i=0;i<8;i++) {
102. pthread\_create(&p\_threads[i], NULL, (void\*)linear\_regression\_thread, &i);
103. pthread\_join(p\_threads[i], NULL);
104. }
106. //printf("best m,c is %lf,%lf with error %lf in direction %d\n",
107. //dm[best\_error\_i], dc[best\_error\_i], best\_error, best\_error\_i);
108. if(best\_error < be) {
109. be = best\_error;
110. **bm = dm[best\_error\_i];**
111. bc = dc[best\_error\_i];
112. } else {
113. minimum\_found = 1;
114. }
115. **}**
116. printf("minimum m,c is %lf,%lf with error %lf**\n**", bm, bc, be);
117. clock\_gettime(CLOCK\_MONOTONIC, &finish);
118. time\_difference(&start, &finish, &time\_elapsed);
119. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed,
120. **(time\_elapsed/1.0e9));**
121. return 0;
122. }
123. point\_t data[] = {
124. {72.12,100.78},{65.40,107.86},{82.27,131.60},{82.31,122.34},
125. **{89.41,121.50},{71.37,113.51},{82.62,112.38},{69.57,102.96},**
126. {65.38,99.27},{84.50,138.85},{87.18,114.17},{73.03,109.21},
127. {67.26,102.06},{72.25,113.23},{61.28,101.59},{41.60,84.24},
128. {40.14,57.03},{15.24,45.58},{61.88,89.90},{34.89,72.77},
129. { 8.91,36.34},{30.45,46.18},{67.93,89.35},{68.82,112.80},
130. **{63.96,99.32},{32.36,56.12},{42.20,63.66},{24.47,60.75},**
131. { 1.96,28.62},{41.42,68.41},{34.49,73.14},{ 8.03,22.13},
132. {80.55,117.79},{85.54,130.80},{68.99,103.13},{99.32,144.79},
133. {91.71,153.61},{71.17,108.40},{85.28,120.11},{99.52,128.68},
134. {13.24,31.67},{ 5.19,40.15},{ 9.84,57.36},{29.42,54.01},
135. **{89.68,126.25},{29.45,41.30},{79.63,132.59},{71.88,107.31},**
136. {20.05,48.38},{40.98,54.11},{56.55,63.61},{77.22,114.17},
137. {63.86,88.10},{92.93,134.84},{56.84,101.20},{34.31,71.18},
138. {93.89,116.43},{38.02,63.78},{61.25,94.71},{71.02,103.42},
139. {95.05,142.82},{96.24,133.50},{19.50,50.92},{41.14,70.59},
140. **{91.49,134.05},{54.05,98.31},{36.59,68.48},{91.14,130.45},**
141. {44.76,88.98},{77.28,138.16},{64.80,96.33},{43.25,70.08},
142. {55.55,95.70},{ 3.77,39.03},{ 3.23,44.69},{86.72,127.42},
143. {84.62,131.54},{26.13,71.24},{61.22,98.22},{53.90,96.07},
144. {64.81,109.35},{91.66,116.79},{53.65,104.81},{38.42,66.16},
145. **{62.33,112.41},{ 7.41,29.86},{41.59,57.59},{56.49,91.60},**
146. {15.94,42.82},{97.46,140.29},{57.17,85.11},{26.94,45.86},
147. {73.14,96.37},{18.61,60.58},{15.69,44.16},{20.79,33.86},
148. {65.02,106.03},{38.09,72.71},{87.15,116.68},{77.45,123.08},
149. {90.47,126.33},{26.80,44.96},{75.94,119.76},{33.83,69.11},
150. **{63.59,103.98},{38.05,72.36},{68.28,110.76},{ 3.34,54.22},**
151. {45.40,92.84},{78.37,113.49},{27.11,46.46},{32.32,68.44},
152. {20.97,30.90},{37.92,75.11},{96.85,130.96},{69.40,95.17},
153. { 3.29,30.06},{64.41,103.44},{15.80,52.64},{61.76,97.79},
154. { 1.62,33.98},{29.03,58.02},{18.74,34.93},{25.41,73.73},
155. **{28.78,65.94},{14.64,50.31},{82.85,133.70},{41.62,90.32},**
156. {99.28,144.95},{90.16,133.18},{40.45,77.72},{ 1.79,50.44},
157. {31.80,62.71},{26.30,40.89},{47.57,83.15},{17.78,44.90},
158. {69.48,93.13},{87.98,126.95},{69.84,106.00},{37.06,61.61},
159. {90.65,133.97},{10.73,46.60},{38.84,79.90},{ 4.75,33.89},
160. **{48.99,89.31},{ 2.51,47.09},{34.99,86.40},{29.79,54.52},**
161. {91.30,133.72},{74.12,122.86},{90.93,141.88},{51.14,89.93},
162. {84.53,142.49},{26.84,58.79},{ 6.95,20.98},{49.80,85.14},
163. {22.82,57.02},{44.08,89.32},{22.28,48.72},{21.12,50.68},
164. {65.69,93.93},{27.84,39.97},{ 1.92,40.39},{ 9.36,33.54},
165. **{88.10,123.02},{18.15,63.84},{21.80,39.76},{64.42,101.03},**
166. { 2.23,22.52},{55.68,99.56},{37.55,87.77},{74.23,104.87},
167. {11.96,37.30},{23.60,45.84},{11.13,34.32},{ 9.05,48.79},
168. {56.11,100.21},{19.31,54.44},{ 6.27,16.17},{64.65,101.39},
169. {50.25,77.59},{69.33,95.12},{47.52,87.79},{28.97,65.98},
170. **{71.56,95.30},{19.71,41.47},{57.66,96.65},{41.07,74.10},**
171. {35.08,79.46},{40.80,87.01},{ 0.31,19.82},{90.78,111.55},
172. {34.39,72.03},{99.97,139.40},{30.86,73.03},{14.37,50.15},
173. { 6.11,42.76},{21.75,80.30},{89.94,127.56},{10.86,42.40},
174. {13.07,42.98},{84.47,147.14},{83.44,132.18},{32.24,63.57},
175. **{66.93,102.41},{34.48,68.96},{ 3.46,22.82},{94.84,130.83},**
176. {49.41,107.26},{71.64,99.82},{47.28,80.62},{39.17,68.77},
177. {58.05,108.35},{69.27,109.81},{47.64,73.34},{34.64,73.15},
178. {22.86,46.34},{37.76,66.19},{ 3.12,39.11},{60.59,111.05},
179. {91.99,122.76},{96.60,138.86},{ 3.58,23.35},{22.81,60.18},
180. **{13.93,21.32},{69.51,106.41},{19.57,43.39},{79.11,115.68},**
181. {80.89,124.36},{44.42,57.78},{33.28,73.04},{21.45,49.88},
182. {70.57,113.77},{45.63,65.60},{55.99,72.21},{21.62,41.47},
183. {61.74,98.99},{ 9.30,29.77},{75.32,106.74},{27.97,73.44},
184. {74.77,115.98},{42.93,82.67},{92.32,138.05},{25.55,64.34},
185. **{ 0.48,23.51},{79.52,111.52},{52.83,70.58},{51.45,87.28},**
186. {62.72,90.41},{ 4.16,40.60},{70.13,115.25},{55.96,97.34},
187. {93.88,154.09},{46.21,90.04},{34.75,51.46},{54.45,89.56},
188. {80.69,129.36},{45.14,73.00},{47.34,85.69},{70.16,118.02},
189. { 4.26,17.14},{61.56,98.04},{15.95,28.56},{74.06,118.48},
190. **{65.29,99.71},{19.08,55.64},{37.82,72.36},{58.22,103.93},**
191. {50.52,82.15},{26.25,60.91},{97.77,123.91},{39.13,68.03},
192. {15.09,41.88},{32.61,61.64},{11.23,22.85},{61.92,98.02},
193. {73.63,126.32},{35.12,54.74},{12.98,42.69},{83.87,128.60},
194. {45.65,78.81},{42.85,90.57},{76.74,117.53},{19.05,49.60},
195. **{69.03,104.16},{23.66,54.97},{52.85,85.94},{82.07,128.27},**
196. {74.77,111.22},{95.04,136.69},{40.49,49.53},{ 4.16,28.40},
197. { 7.69,51.29},{29.37,80.82},{86.06,122.19},{ 3.92,23.24},
198. {62.76,108.89},{27.12,54.24},{10.24,33.84},{79.86,107.97},
199. {57.09,85.27},{10.29,54.38},{53.50,82.98},{12.83,50.29},
200. **{ 2.09,13.69},{88.73,135.16},{42.72,87.10},{40.20,91.88},**
201. {40.10,76.49},{80.22,133.65},{57.55,93.99},{29.34,69.08},
202. { 2.90,41.26},{44.60,82.03},{47.93,89.05},{98.17,123.11},
203. {17.21,45.91},{42.37,79.83},{90.89,119.42},{ 7.81,36.64},
204. {76.14,123.86},{47.79,83.40},{95.27,144.30},{44.13,98.20},
205. **{19.97,37.36},{90.66,131.96},{75.41,117.80},{57.14,107.91},**
206. {25.92,41.69},{90.86,130.36},{44.78,79.02},{23.00,29.10},
207. {91.67,118.13},{26.55,51.18},{41.60,74.91},{ 0.39, 6.79},
208. {86.31,102.08},{20.43,37.80},{ 5.39,28.65},{12.63,24.33},
209. {22.60,42.79},{ 1.77,14.54},{74.10,113.64},{54.46,87.67},
210. **{18.64,49.32},{93.97,116.30},{42.62,87.04},{13.37,30.16},**
211. {74.50,104.62},{18.28,67.85},{76.98,107.84},{25.89,57.35},
212. {13.52,42.87},{61.26,97.78},{ 5.97,31.34},{91.99,137.43},
213. {20.38,58.23},{ 9.59,31.56},{79.41,126.40},{89.90,134.36},
214. {73.18,111.44},{61.51,111.41},{99.96,147.82},{72.55,113.52},
215. **{66.21,110.93},{36.47,59.41},{65.58,93.39},{24.93,51.71},**
216. {58.00,95.89},{49.83,83.52},{53.35,89.98},{83.97,129.85},
217. {57.33,106.86},{53.94,98.13},{98.02,144.26},{47.28,72.52},
218. {45.48,100.70},{80.69,147.66},{96.14,140.01},{82.69,120.80},
219. {79.73,136.89},{11.42,27.51},{88.91,138.59},{25.53,51.26},
220. **{ 2.49,37.14},{63.89,93.28},{90.96,138.02},{15.27,53.03},**
221. {25.39,51.31},{31.77,55.54},{88.25,124.46},{67.66,108.26},
222. {90.23,112.02},{17.40,43.85},{78.38,137.07},{96.28,149.45},
223. {77.38,120.54},{56.49,107.27},{99.00,141.67},{36.35,58.18},
224. {97.41,132.64},{15.03,48.28},{42.48,81.20},{62.95,105.32},
225. **{99.76,147.11},{85.18,140.95},{99.23,131.84},{21.09,44.44},**
226. {45.12,75.22},{80.36,119.71},{61.37,84.74},{82.64,128.58},
227. {70.34,108.16},{83.63,116.26},{47.73,67.57},{17.56,48.42},
228. {23.26,42.12},{41.81,82.17},{18.48,33.63},{39.11,70.14},
229. {84.20,123.97},{67.20,113.97},{52.74,87.79},{81.66,131.54},
230. **{45.90,93.69},{20.82,34.77},{86.35,122.38},{78.93,106.82},**
231. {10.56,44.66},{51.20,104.61},{93.79,131.97},{15.71,43.06},
232. {99.16,156.47},{90.70,135.27},{41.85,77.91},{73.41,106.66},
233. {57.51,108.55},{53.06,115.27},{25.72,67.45},{ 8.03,27.74},
234. {57.91,101.56},{35.87,57.47},{98.33,145.81},{50.96,76.84},
235. **{57.86,102.10},{17.21,44.21},{95.62,154.59},{76.92,114.77},**
236. {25.32,60.66},{43.60,68.34},{42.68,73.98},{60.36,84.81},
237. { 9.06,42.91},{ 4.16,18.44},{54.14,97.87},{ 4.87,35.92},
238. {75.38,112.62},{41.37,68.92},{88.16,163.96},{16.79,41.87},
239. { 9.77,40.62},{69.66,125.12},{70.35,118.66},{71.99,97.87},
240. **{63.66,111.29},{ 2.01,19.46},{64.63,122.89},{48.39,84.19},**
241. {28.15,64.69},{46.17,83.91},{25.12,45.94},{82.23,118.70},
242. {57.69,95.98},{24.42,62.91},{15.81,35.58},{75.28,106.87},
243. {95.74,133.25},{67.78,107.42},{80.89,128.72},{10.39,38.37},
244. {15.31,35.73},{61.45,110.46},{11.15,44.99},{30.80,63.26},
245. **{84.29,122.39},{29.17,47.34},{80.68,138.44},{81.17,117.86},**
246. { 8.47,32.78},{41.26,74.09},{43.50,71.18},{34.48,68.61},
247. {30.63,68.05},{88.63,137.28},{71.56,116.97},{21.03,39.12},
248. {88.20,116.24},{ 8.52,30.24},{95.79,137.27},{78.66,104.62},
249. {72.44,94.21},{71.60,106.34},{72.11,114.18},{34.50,59.18},
250. **{22.85,60.95},{18.43,40.91},{69.24,119.69},{91.84,142.06},**
251. {34.41,69.95},{95.06,136.92},{67.93,100.93},{46.96,71.82},
252. {63.92,102.14},{ 1.62,29.66},{95.24,133.60},{43.10,80.88},
253. {21.83,73.25},{35.01,62.42},{20.05,55.19},{18.64,45.92},
254. {40.28,75.26},{34.54,63.38},{84.74,117.68},{90.38,144.87},
255. **{ 9.91,24.87},{62.97,102.14},{34.40,79.20},{67.34,89.48},**
256. {48.53,85.13},{24.57,51.59},{81.95,117.78},{22.23,49.77},
257. {75.86,125.20},{60.45,99.78},{19.93,35.57},{48.62,78.46},
258. {88.49,120.71},{13.33,40.67},{52.03,93.38},{38.43,80.28},
259. { 2.56,17.00},{18.39,58.10},{58.81,88.08},{75.76,96.69},
260. **{69.78,98.83},{96.47,146.81},{47.32,79.89},{21.90,46.54},**
261. {52.39,83.38},{75.49,107.96},{50.14,80.51},{41.54,73.80},
262. {76.07,117.48},{27.00,73.59},{81.59,122.88},{21.74,39.55},
263. {60.05,105.04},{75.68,102.72},{40.41,79.01},{ 0.32,24.82},
264. {50.06,106.14},{98.69,139.50},{64.17,109.26},{42.74,78.53},
265. **{39.52,71.78},{55.14,97.37},{25.19,39.08},{99.31,142.63},**
266. {67.50,91.86},{90.92,152.17},{81.99,129.38},{77.28,124.08},
267. {29.38,69.15},{ 3.81,41.93},{ 9.72,41.83},{25.75,53.09},
268. {57.28,85.11},{69.50,116.90},{20.00,51.46},{63.00,72.32},
269. {67.06,102.20},{37.85,64.86},{81.40,114.28},{13.32,58.41},
270. **{67.21,103.77},{63.73,109.66},{91.43,141.66},{54.83,88.07},**
271. {68.03,112.67},{ 0.51,27.76},{ 2.17,38.05},{36.26,66.58},
272. {72.67,116.52},{98.28,136.37},{85.27,128.64},{90.26,136.47},
273. {60.31,95.24},{32.77,58.94},{ 3.52,24.75},{15.98,45.49},
274. {94.25,145.90},{ 8.13,29.89},{61.13,81.38},{44.14,77.64},
275. **{63.53,100.35},{49.35,97.92},{ 4.98,32.12},{25.53,57.45},**
276. { 8.63,41.62},{24.23,56.27},{93.30,137.92},{43.72,71.72},
277. {54.15,89.12},{ 3.42,36.34},{57.75,85.68},{51.90,87.74},
278. {85.14,137.82},{99.27,173.87},{82.53,124.94},{15.38,44.42},
279. {66.66,108.56},{64.12,99.41},{39.08,73.77},{25.42,58.25},
280. **{ 1.29,36.39},{98.72,148.84},{70.09,112.06},{ 8.51,27.00},**
281. {85.92,124.74},{88.32,127.04},{51.79,74.58},{36.46,62.45},
282. {49.29,85.33},{14.06,30.58},{24.83,34.82},{42.85,87.06},
283. {34.47,76.96},{59.16,90.44},{ 1.02,32.32},{61.80,108.22},
284. {72.52,95.83},{65.40,99.49},{53.32,93.79},{74.22,117.61},
285. **{53.86,88.31},{39.84,80.11},{79.28,117.86},{34.57,76.73},**
286. {21.69,55.55},{99.87,129.34},{72.12,108.86},{75.08,106.64},
287. {70.71,106.00},{18.35,67.45},{37.42,66.71},{ 0.70, 9.02},
288. {56.79,86.75},{74.04,100.45},{53.40,82.23},{42.13,70.45},
289. {82.43,123.55},{91.65,131.55},{94.99,153.70},{62.14,84.17},
290. **{99.71,151.07},{33.24,73.77},{48.87,76.91},{68.57,118.95},**
291. {14.28,46.22},{18.17,41.01},{95.93,133.32},{ 5.06,33.23},
292. {57.58,95.47},{18.71,39.10},{90.19,136.73},{26.98,50.08},
293. {11.36,26.14},{62.70,98.59},{49.32,80.54},{99.97,149.27},
294. {83.40,132.00},{25.30,48.62},{79.25,117.83},{81.09,109.23},
295. **{31.46,51.02},{14.26,32.26},{33.53,52.63},{ 9.42,47.16},**
296. {67.40,109.90},{18.56,32.79},{34.51,75.14},{49.00,77.38},
297. {15.69,50.80},{23.09,40.32},{32.03,67.86},{13.60,40.35},
298. {19.21,60.16},{78.56,111.57},{80.72,131.02},{50.19,79.64},
299. {55.60,81.78},{ 6.37,43.37},{42.78,74.85},{60.48,113.67},
300. **{44.44,89.27},{54.02,90.24},{73.51,101.74},{16.41,56.73},**
301. {70.94,104.90},{32.03,66.91},{13.12,49.71},{50.16,85.64},
302. {41.31,68.88},{69.25,123.25},{24.97,69.28},{40.80,86.30},
303. {32.28,67.01},{90.77,142.80},{66.77,104.70},{24.06,56.12},
304. {49.16,89.52},{46.10,95.56},{51.79,94.01},{56.11,100.66},
305. **{88.49,126.71},{ 1.28,21.35},{35.55,64.10},{18.79,29.74},**
306. { 5.40,40.02},{92.32,129.89},{21.13,47.05},{ 5.14,32.16},
307. {60.89,104.41},{43.45,76.07},{98.91,160.53},{99.31,155.80},
308. {74.71,121.53},{62.33,98.98},{58.66,101.10},{51.51,93.03},
309. {51.69,90.42},{19.47,31.22},{85.75,108.87},{64.20,100.48},
310. **{96.60,142.66},{67.99,102.48},{68.37,120.07},{29.81,44.77},**
311. {96.55,142.74},{30.59,43.25},{73.94,108.44},{49.77,88.88},
312. {59.48,98.21},{41.21,61.86},{38.63,83.41},{86.98,140.40},
313. {93.34,134.69},{87.92,119.52},{40.93,61.87},{ 2.43,30.68},
314. {50.74,71.81},{37.13,52.43},{ 1.50,22.18},{99.06,143.48},
315. **{ 1.67,27.67},{ 0.18,10.50},{54.13,77.05},{46.19,88.91},**
316. {91.13,144.49},{ 8.95,28.33},{85.69,122.61},{50.30,95.60},
317. {48.63,103.49},{67.99,100.19},{69.21,112.13},{11.26,34.99},
318. {25.78,58.73},{84.35,112.36},{46.80,79.68},{69.54,117.99},
319. {40.30,74.33},{79.97,118.95},{23.28,55.71},{32.62,78.92},
320. **{21.86,37.01},{ 5.07,22.57},{94.41,146.15},{40.14,60.81},**
321. {95.80,125.35},{91.34,131.68},{72.55,113.56},{40.13,71.59},
322. {98.06,145.27},{90.55,144.08},{71.26,121.81},{33.85,71.13},
323. {85.74,142.63},{57.93,91.78},{ 7.63,39.30},{83.72,128.26},
324. {10.89,46.78},{39.79,66.98},{98.84,146.32},{84.62,123.91},
325. **{23.16,31.94},{86.36,134.79},{44.19,63.74},{ 0.39,24.19},**
326. {64.22,96.97},{66.47,103.78},{ 1.73,17.52},{22.25,36.77},
327. {31.88,59.39},{15.60,30.03},{16.08,41.91},{83.11,129.19},
328. {72.61,122.52},{19.02,41.06},{56.90,87.53},{65.85,97.02},
329. {81.40,120.35},{64.90,104.44},{73.35,119.00},{ 8.49,40.31},
330. **{31.20,65.32},{28.29,75.05},{72.51,120.90},{20.42,48.84},**
331. {71.46,111.59},{33.98,50.46},{72.48,111.29},{75.56,113.00},
332. {58.65,95.16},{23.66,44.95},{95.08,139.46},{80.12,115.20},
333. {67.77,101.97},{56.06,99.08},{99.03,138.47},{48.26,74.79},
334. {25.95,39.30},{85.20,137.70},{69.31,104.19},{86.19,122.91},
335. **{37.99,87.47},{72.06,116.90},{ 5.66,28.92},{27.77,52.05},**
336. {31.89,60.32},{18.01,48.92},{37.21,65.49},{73.76,107.20},
337. { 0.32,-0.71},{93.75,133.48},{69.11,109.63},{11.01,55.84},
338. {43.48,73.99},{20.76,57.44},{75.50,105.00},{98.74,150.46},
339. {40.75,90.93},{61.67,103.30},{93.48,155.96},{35.52,61.62},
340. **{32.30,78.52},{28.92,49.61},{60.97,87.11},{13.59,47.58},**
341. { 9.43,26.07},{58.00,107.90},{99.86,151.90},{34.01,57.82},
342. {39.02,59.14},{33.64,74.99},{ 2.28,20.21},{55.00,90.93},
343. {55.77,85.94},{79.17,134.03},{63.16,106.70},{17.58,32.28},
344. {24.29,34.68},{83.91,132.35},{96.44,129.86},{61.95,93.66},
345. **{14.86,25.10},{15.53,33.29},{15.69,42.47},{80.60,126.11},**
346. {16.01,46.33},{26.54,74.55},{ 2.67,37.10},{74.63,96.98},
347. {38.06,59.99},{56.59,96.87},{78.88,120.95},{87.56,121.75},
348. {73.54,119.27},{16.84,44.09},{44.24,89.36},{76.02,123.64},
349. {98.41,115.45},{12.11,48.19},{30.70,60.41},{55.51,100.49},
350. **{ 0.26,37.11},{83.43,124.44},{49.92,111.30},{65.55,99.48},**
351. {77.61,119.44},{62.44,95.52},{21.80,61.06},{20.99,60.54},
352. {93.10,129.45},{54.96,91.05},{10.22,48.48},{66.77,108.83},
353. {40.83,87.14},{13.54,35.77},{31.44,62.92},{79.69,110.30},
354. {67.07,100.59},{28.81,78.71},{52.95,97.30},{39.89,81.67},
355. **{58.79,75.89},{34.35,51.29},{38.03,64.97},{87.87,130.19},**
356. {39.73,52.43},{ 1.64,31.22},{91.15,147.58},{54.08,101.10},
357. {53.53,74.54},{54.24,104.47},{15.04,51.28},{79.06,114.59},
358. {93.83,138.37},{94.89,122.18},{52.63,86.22},{27.83,68.05},
359. {54.51,94.07},{23.83,58.00},{86.88,141.66},{10.42,31.81},
360. **{55.43,84.31},{45.04,85.30},{95.69,121.78},{17.28,35.32},**
361. { 3.17,33.76},{51.61,69.81},{27.37,64.13},{88.92,160.98},
362. {31.40,64.46},{33.35,59.91},{82.48,128.89},{50.46,98.13},
363. {78.73,113.68},{70.08,115.27},{98.65,142.28},{ 9.15,50.95},
364. {16.74,35.73},{32.92,72.02},{ 1.29,18.94},{75.79,123.45},
365. **{32.94,59.92},{61.72,81.50},{42.39,91.90},{70.15,108.81},**
366. { 2.90,29.10},{59.68,87.41},{69.85,108.66},{71.21,107.81},
367. {24.09,46.47},{44.51,76.59},{ 7.30,34.83},{58.93,99.24},
368. { 1.24,22.60},{84.27,132.21},{54.11,87.19},{39.18,75.93},
369. {90.81,155.72},{67.68,88.19},{67.14,84.53},{53.98,86.47},
370. **{67.28,106.68},{ 8.49,36.74},{34.96,62.55},{59.01,82.94},**
371. {64.78,101.77},{66.24,110.82},{75.81,131.28},{62.82,76.02},
372. {73.95,116.37},{20.40,38.76},{45.06,84.65},{47.64,82.81},
373. {30.85,64.41},{77.10,112.67},{ 8.12,32.76},{39.56,53.41}
374. };

**Explanation for multithread linear regression:**

The main motive behind this code is to find out the minimum m and c value with best error. M and c are the slope or gradient of line and intercept of the straight line. Slope deals with direction and line’s steepness where intercepts are the point where it crosses the axis. There are intercepts x- intercept and y-intercept. We tried to find out the value of m and c with optimal root mean square error by following steps:

Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively Time elapsed is calculated by subtracting finish time and start time after getting the both the time. We have declared the bm, bc and best\_error with values 1.3, 10 and 999999999. rms\_error() with parameter m and c is the function to find the error of the straight line. Rms error can be calculated by square rooting all sum of squares of error divided by number of errors. Mathematically, it can be represented by:

Root Mean Square error=

Residual error is calculated by the difference between observed value and predicted value. Before finding the RMS error we must find the residual error to find the minimum error.

Residual error = observed value – predicted value

Inside rms\_error() function, another function residual\_error() function with parameter double x, double y, double m, double c. The value of x and y are taken from the data provided by the university to us. The residual error is calculated by the formula e = (m \* x) + c – y and square of e is returned to the function rms\_error(). In similar fashion, eight threads using the for loop. Inside for loop, we have created the thread with pthread\_create() function which takes four parameters. First parameter is identifier for new thread, second parameter is thread attribute object, third parameter is start routine and fourth parameter is single argument passed to third parameter that is start routine. In our case we have &threads[t], NULL, regression\_thread, &t. t is set to zero first and increased by one in each loop. Array of dm[] and dc[] with 8 values is initialized and best error is calculated for different value of m and c by comparing the error with best error, Best error is set to 999999999 only to compare the value because the error will not come this high in any case. First the value of best error will be compared with 999999999 and it is obvious that the new value will be always smaller than that and best error will be replaced by the value. Later for all the best error, the best error value is replaced when smaller value comes which helps us to find the minimum value of error. pthread\_join is library function which takes thread and value\_ptr as argument. In our case, threads[t] is the argument for the thread and value\_ptr is set to NULL. Then only the minimum value of m, c and error is displayed in the terminal by using print statement. The total time taken by the program is calculated by the function named time\_difference() function and displayed in the terminal.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (multithread) |
| 1 | 52689676ns or 0.052689676s | 223212433ns or 0.223212433s |
| 2 | 52694668ns or 0.052694668s | 223212433ns or 0.223212433s |
| 3 | 52739608ns or 0.052739608s | 224028255ns or 0.224028255s |
| 4 | 52767176ns or 0.052767176s | 226202530ns or 0.226202530s |
| 5 | 52658329ns or 0.052658329s | 223020605ns or 0.223020605s |
| 6 | 52725256ns or 0.052725256s | 232051604ns or 0.232051604s |
| 7 | 53270258ns or 0.053270258s | 223344587ns or 0.223344587s |
| 8 | 52675616ns or 0.052675616s | 229275053ns or 0.229275053s |
| 9 | 53785147ns or 0.053785147s | 227676005ns or 0.227676005s |
| 10 | 52968744ns or 0.052968744s | 230895031ns or 0.230895031s |
|  | Mean Running time for original program:  52897447.8ns or 0.52897447s | Mean Running time for multithread program: 226291853.6 ns or 0.2262918536s |

Table 4 Running time for multithread

**Write a short analysis of the results.**

The Mean running time for original program is 52897447.8ns or 0.52897447s without using any parallel computing. Multithread program took around 226291853.6 ns or 0.2262918536s because we used eight threads in single program. It took less time than original program because original program uses only one thread. The multithread program is = 2.33757628295 ≈ 2 times faster than the original program.

# 2. CUDA

## 2.1 Password Cracking

**Code for CUDA Password Cracking:**

1. #include <stdio.h>
2. #include <cuda\_runtime\_api.h>
3. #include <time.h>
4. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**5.**

1. *Compile and run with:*
2. *nvcc -o PassBishal PassBishal.cu*
3. *./PassBishal*

9.  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

1. *This function returns 1 if the attempt at cracking the password is*
2. ***identical to the plain text password string stored in the program.***
3. *Otherwise,it returns 0.*
4. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

18.

1. \_\_device\_\_ int is\_a\_match(char \*attempt) {
2. **char password1[] = "BI2344";**
3. char password2[] = "SH5699";
4. char password3[] = "AL0874";
5. char password4[] = "KH0023";

24.

**25.**

1. char \*a = attempt;
2. char \*b = attempt;
3. char \*c = attempt;
4. char \*d = attempt;
5. **char \*p1 = password1;**
6. char \*p2 = password2;
7. char \*p3 = password3;
8. char \*p4 = password4;

34.

1. **while(\*a == \*p1) {**
2. if(\*a == '**\0**')
3. {
4. printf("Password: %s**\n**",password1);
5. **break**;
6. **}**

41.

1. a++;
2. p1++;

44. }

**45.**

1. while(\*b == \*p2) {
2. if(\*b == '**\0**')
3. {
4. printf("Password: %s**\n**",password2);
5. **break;**
6. }

52.

1. b++;
2. p2++;
3. **}**

56.

1. while(\*c == \*p3) {
2. if(\*c == '**\0**')
3. {
4. **printf("Password: %s\n",password3);**
5. **break**;
6. }

63.

1. c++;
2. **p3++;**
3. }

67.

1. while(\*d == \*p4) {
2. if(\*d == '**\0**')
3. **{**
4. printf("Password: %s**\n**",password4);
5. return 1;
6. }

74.

1. **d++;**
2. p4++;
3. }
4. return 0;

79.

**80. }**

81.

* 1. \_\_global\_\_ void kernel() {
  2. char i1,i2,i3,i4;

84.

**85. char password[7];** 86. password[6] = '**\0**';

87.

* 1. int i = blockIdx.x+65;
  2. int j = threadIdx.x+65;
  3. **char firstMatch = i;** 91. char secondMatch = j;

92.

* 1. password[0] = firstMatch;
  2. password[1] = secondMatch;
  3. **for(i1='0'; i1<='9'; i1++){**
  4. for(i2='0'; i2<='9'; i2++){
  5. for(i3='0'; i3<='9'; i3++){
  6. for(i4='0'; i4<='9'; i4++){ 99. password[2] = i1;
  7. **password[3] = i2;**
  8. password[4] = i3;
  9. password[5] = i4;
  10. if(is\_a\_match(password)) {
  11. }
  12. **else {**
  13. *//printf("tried: %s\n", password);*
  14. }
  15. }
  16. }
  17. **}**
  18. }

112. } 113.

1. int time\_difference(struct timespec \*start,
2. **struct timespec \*finish,**
3. long long int \*difference) {
4. long long int ds = finish->tv\_sec - start->tv\_sec;
5. long long int dn = finish->tv\_nsec - start->tv\_nsec;
6. if(dn < 0 ) {
7. **ds--;**
8. dn += 1000000000;
9. }
10. \*difference = ds \* 1000000000 + dn;
11. return !(\*difference > 0);
12. **}** 126.

127.

128. int main() {

129.

1. **struct timespec start, finish;**
2. long long int time\_elapsed; 132. clock\_gettime(CLOCK\_MONOTONIC, &start);

133.

134. kernel <<<26,26>>>(); **135. cudaThreadSynchronize();** 136.

1. clock\_gettime(CLOCK\_MONOTONIC, &finish);
2. time\_difference(&start, &finish, &time\_elapsed);
3. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed, (time\_elapsed/1.0e9));

**140.**

1. return 0;
2. }

**Explanation CUDA Password Cracking:**

Password cracking with the help of CUDA is done to parallel compute and speed up to solve the problem by using graphical processing unit and the resulting time is calculated. Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. CLOCK\_MONOTIC is used to get the absolute time right after the execution of program. CLOCK\_MONOTONIC is used instead of CLOCK\_REALTIME because CLOCK\_MONOTONIC is not affected by the system data and time. Time elapsed is calculated by subtracting finish time and start time after getting the both the time. The CUDA Password Cracking is done in sequential manner. First the declaration and allocation of both host memory (CPU) and device memory (GPU) was done and the host data was initialized. After that the data was transferred to device (GPU) from the host (CPU). Then host execute all the program and finally the result was transferred to the host from device. The program is divided into number of thread and blocks that run independently of each other. Our program uses kernel <<26,26>> function which will execute in 26 thread blocks consisting of 26 thread. Triple Angular bracket is also known as kernel launch. The kernel function will be called up only one time and \_\_global\_\_ function is used which will execute on GPU and called from CPU. Inside \_\_global\_\_ function named kernel, four loops variable i1, i2, i3, i4 with char datatypes are declared. Also password array is introduced and last value of password is set to \0. Since each block handles different array elements, each array is indexed using blockIdx.x. Similarly threadIdx.x is used to access the thread index. Then program will run on multiple. 65 is added in blockIdx.x to make the capital letter. Our passwords contain plain password with two initial capital alphabet and four digits. So, the initial two letters must be extracted from blockIdx.x+ 65 and threadIdx.x +65 respectively. The other 4 digits must be put on a loop to find out the desired output. So i1, i2, i3 and i4 are all put in for loop starting from 0 to 9. To all the combination, the result is checked with the given password by using is\_a\_match functipn which take password array as parameter. Inside is\_a\_match function, password is pointed by the attempt pointer and all four combination are stored in four pointer named \*a, \*b, \*c, \*d and our actual password are stored in pointer named \*p1, \*p2, \*p3, \*p4. Finally all combination password are checked with our actual password. Four passwords are printed in the terminal only if combination pointer and out actual password pointer matched.

Before calculating the time taken by the program cudaThreadSynchronize() is called to block until all the previous requested tasks have been completed by the system. After all of this finish time is taken to calculate the total time taken by the program with the help of time\_difference() function.

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

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Running time in nano second | Running time in second | Mean Running Time |
| 1 | 54935099ns | 0. 054935099s | 54975897.3ns or 0.0549758973s |
| 2 | 54777348ns | 0. 054777348s |
| 3 | 56374022ns | 0. 056374022s |
| 4 | 55138440ns | 0. 055138440s |
| 5 | 55051498ns | 0. 055051498s |
| 6 | 55228337ns | 0. 055228337s |
| 7 | 54882557ns | 0. 054882557s |
| 8 | 54411646ns | 0.054411646s |
| 9 | 54220145ns | 0. 054220145s |
| 10 | 54739881ns | 0. 054739881s |

Table 5 Running time for CUDA Password cracking

**The comparisons table of original program with CUDA version:**

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (multithread) |
| 1 | 516.887633220s | 0. 054935099s |
| 2 | 506.811146889s | 0. 054777348s |
| 3 | 508.927593166s | 0. 056374022s |
| 4 | 507.269538405s | 0. 055138440s |
| 5 | 507.184282818s | 0. 055051498s |
| 6 | 508.754546574s | 0. 055228337s |
| 7 | 508.737758971s | 0. 054882557s |
| 8 | 509.791337001s | 0.054411646s |
| 9 | 517.265658095s | 0. 054220145s |
| 10 | 506.752829585s | 0. 054739881s |
|  | Mean Running time for original program: 509838232472.4ns or 509.8382324724s | Mean Running time for CUDA program: 54975897.3ns or 0.0549758973s |

Table 6 Comparision between original and multithread password cracking

**Write a short analysis of the results:**

CUDA is developed by Nvidia to improve the speed and pace by using the Graphical Processing unit. Other competitor like OpenCL and AMD are also struggling to increase the performance still CUDA is dominant to other APIs. After this statement anyone can say the time would be much less than the original time. The mean running time for original program: 509838232472.4ns or 509.8382324724s or 8.49730387454 minutes. We have already used multithreading posix which gives the running time of 258.5557073435 s or 4.309261789058334 minutes as two threads are used in the program. We got surprised after seeing the mean Running time for CUDA program is found to be only 54975897.3ns or 0.0549758973s or 0.000916264955 minutes. The program is = 9273.85013273 times faster than the original program and = 4703.07389314 times faster than the posix thread version. This was the reason why CUDA is being used in the field of deep learning and it is being continuously optimized by NVIDIA.

## 2.2 Image Processing

**2.2.1 Code for CUDA Image Processing:**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <GL/glut.h>
5. **#include <GL/gl.h>**
6. #include <malloc.h>
7. #include <signal.h>
8. #include <cuda\_runtime\_api.h>
9. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
10. ***The variable names and the function names of this program is same as provided by the university.***
11. *The added variable and function are the only changes made to this program.*
13. *Compile with:*
14. *nvcc -o Cudaimg Cudaimg.cu -lglut -lGL*
16. *./Cudaimg > results.txt*


20. ***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***
21. #define width 100
22. #define height 72
24. unsigned char results[width \* height];
25. **unsigned char image[] = {255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
26. **255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,255,255,**
27. **0,255,255,255,0,0,0,0,255,255,0,0,0,0,0,0,255,255,255,**
28. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
29. **255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
30. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,**
31. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,**
32. **255,0,0,255,255,0,0,255,255,0,255,255,0,0,255,0,0,255,255,**
33. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
34. **255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,**
35. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
36. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,**
37. **255,255,255,0,0,255,255,0,0,255,255,0,255,0,0,255,255,0,0,**
38. **255,255,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,**
39. **255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,**
40. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
41. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,**
42. **255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,0,0,0,**
43. **0,255,255,255,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,**
44. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,**
45. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
46. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
47. **255,255,0,255,255,255,255,255,0,0,255,255,255,0,0,0,255,255,0,**
48. **0,255,0,0,0,0,255,255,255,0,0,255,255,0,255,255,255,255,255,**
49. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
50. **255,255,255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
51. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
52. **255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,**
53. **0,0,0,255,0,0,255,255,0,0,0,255,255,255,255,0,255,255,255,**
54. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
55. **255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,**
56. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
57. **255,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,**
58. **0,0,0,255,255,255,0,0,255,255,0,0,255,0,0,0,255,255,255,**
59. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
60. **255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,**
61. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
62. **255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,0,**
63. **0,255,255,255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,**
64. **0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
65. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,**
66. **0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
67. **0,0,0,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,255,**
68. **255,255,255,255,255,0,255,255,255,255,255,0,255,255,255,255,255,0,0,**
69. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
70. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
71. **255,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,**
72. **0,0,0,0,0,0,0,0,0,255,255,0,0,255,255,255,0,0,0,**
73. **255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,255,**
74. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
75. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
76. **255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,**
77. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,**
78. **255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,**
79. **255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
80. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
81. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
82. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
83. **255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,**
84. **255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
85. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
86. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
87. **0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
88. **0,0,0,0,255,255,255,255,255,0,255,0,0,255,255,0,0,255,255,**
89. **255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
90. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
91. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
92. **0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,**
93. **0,0,0,0,0,0,0,0,0,255,0,0,255,255,0,0,0,0,255,**
94. **255,255,0,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,**
95. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
96. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
97. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
98. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
99. **255,0,0,0,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,**
100. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
101. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
102. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
103. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
104. **255,255,255,255,255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,**
105. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
106. **255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
107. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
108. **0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,**
109. **0,0,0,0,0,255,255,255,255,255,255,0,0,0,255,255,255,255,255,**
110. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
111. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
112. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
113. **255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,**
114. **0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,255,255,0,255,**
115. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
116. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
117. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
118. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,**
119. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
120. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
121. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,**
122. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
123. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
124. **0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
125. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
126. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
127. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
128. **0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,255,255,255,**
129. **255,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,**
130. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,**
131. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
132. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
133. **0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,255,255,0,**
134. **0,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,0,0,0,**
135. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
136. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
137. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
138. **0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,0,**
139. **255,255,255,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,255,**
140. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
141. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
142. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
143. **0,0,0,0,0,0,0,255,255,0,0,255,255,255,255,0,0,255,255,**
144. **255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,255,255,0,0,**
145. **0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
146. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
147. **255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
148. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,**
149. **255,0,0,255,255,255,255,255,255,0,255,255,255,255,0,0,255,255,255,**
150. **255,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,**
151. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
152. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
153. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,**
154. **255,0,0,255,255,255,0,0,255,255,255,255,255,255,0,255,255,255,255,**
155. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,**
156. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
157. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
158. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
159. **255,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,**
160. **255,255,255,255,255,0,0,0,0,0,255,255,255,0,0,0,0,0,0,**
161. **0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
162. **255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,**
163. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
164. **0,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,**
165. **255,255,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,**
166. **0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,**
167. **0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
168. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,**
169. **255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,0,**
170. **255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,**
171. **255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,**
172. **0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,**
173. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
174. **255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,0,0,0,255,**
175. **255,255,255,0,0,255,255,255,255,255,255,255,0,255,255,255,0,0,255,**
176. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,255,**
177. **255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,**
178. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
179. **255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,0,255,255,255,**
180. **255,0,0,0,0,255,255,255,255,0,255,255,255,255,255,255,0,0,255,**
181. **255,255,255,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
182. **0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
183. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
184. **0,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,**
185. **255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,**
186. **255,255,0,0,255,255,255,255,0,0,0,255,255,255,255,255,0,0,0,**
187. **0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,**
188. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
189. **0,0,0,0,255,255,255,255,0,255,255,255,0,0,0,255,255,255,0,**
190. **0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,**
191. **0,0,255,255,255,255,255,0,0,0,255,255,255,0,0,0,0,0,0,**
192. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
193. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
194. **0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,**
195. **0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,**
196. **0,0,255,255,255,0,0,255,255,255,255,255,0,0,0,255,255,255,0,**
197. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
198. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
199. **0,0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,**
200. **255,255,255,255,0,255,255,255,255,0,0,0,0,255,255,255,255,255,255,**
201. **255,255,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,**
202. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
203. **0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
204. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
205. **255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,0,0,0,255,**
206. **255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,0,255,**
207. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
208. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
209. **0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,255,255,**
210. **255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,**
211. **255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,**
212. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
213. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,**
214. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,**
215. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,**
216. **255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,255,255,**
217. **255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
218. **0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,**
219. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
220. **255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
221. **0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,**
222. **0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
223. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,**
224. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
225. **0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,**
226. **0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,**
227. **0,255,255,255,255,0,0,0,255,255,0,0,0,0,0,0,0,0,0,**
228. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
229. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
230. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,0,**
231. **0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,**
232. **255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,**
233. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,**
234. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
235. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
236. **255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,0,0,0,255,**
237. **255,255,255,255,255,255,255,255,255,0,0,0,255,0,0,0,0,0,0,**
238. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
239. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
240. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
241. **0,0,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,**
242. **255,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
243. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
244. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
245. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
246. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,255,**
247. **255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,0,0,0,**
248. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
249. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
250. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
251. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,255,**
252. **255,255,255,0,0,255,255,255,255,255,255,0,0,0,0,255,255,255,0,**
253. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
254. **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,**
256. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
257. **255,255,0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,0,**
258. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
259. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,**
260. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
261. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
262. **0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,**
263. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
264. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
265. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
266. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
267. **0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,**
268. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
269. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,**
270. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
271. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
272. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
273. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
274. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
275. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
276. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
277. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
278. **0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
279. **0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
280. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
281. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
282. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
283. **0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,**
284. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,**
285. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
286. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
287. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
288. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
289. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
290. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
291. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
292. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
293. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
294. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,**
295. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
296. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
297. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
298. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
299. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
300. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
301. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
302. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
303. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
304. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
305. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
306. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
307. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
308. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
309. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
310. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
311. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
312. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
313. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
314. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
315. **0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
316. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
317. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
318. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
319. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
320. **0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,**
321. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
322. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
323. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
324. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
325. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
326. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
327. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
328. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
329. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
330. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,**
331. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
332. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
333. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
334. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
335. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
336. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
337. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
338. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
339. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
340. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
341. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
342. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
343. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
344. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
345. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
346. **0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
347. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
348. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
349. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
350. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
351. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
352. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
353. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
354. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
355. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
356. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
357. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
358. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
359. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
360. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
361. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
362. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
363. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
364. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
365. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
366. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
367. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
368. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
369. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
370. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
371. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
372. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
373. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
374. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
375. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
376. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
377. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
378. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
379. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
380. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
381. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
382. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
383. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
384. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
385. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
386. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
387. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
388. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
389. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
390. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
391. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
392. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
393. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
394. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
395. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
396. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
397. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
398. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
399. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
400. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
401. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
402. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
403. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0**
404. **};**\_\_global\_\_ void detect\_edges(unsigned char \*in, unsigned char \*out) {
405. int i = (blockIdx.x \* 72) + threadIdx.x;
406. int x, y; // the pixel of interest
407. int b, d, f, h; // the pixels adjacent to x,y used for the calculation
408. **int r; // the result of calculate**
409. y = i / width;
410. x = i - (width \* y);
411. **if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {**
412. out[i] = 0;
413. }
414. } else {
415. b = i + width;
416. d = i - 1;
417. **f = i + 1;**
418. h = i - width;
419. r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)+ (in[h] \* -1);
420. if (r > 0) { // if the result is positive this is an edge pixel
421. out[i] = 255;} else {
422. out[i] = 0;
423. **}**
424. }
425. **void tidy\_and\_exit() {**
426. exit(0);
427. }
428. void sigint\_callback(int signal\_number){
429. **printf("\nInterrupt from keyboard\n");**
430. tidy\_and\_exit();
431. }
432. static void display() {
433. **glClear(GL\_COLOR\_BUFFER\_BIT);**
434. glRasterPos4i(-1, -1, 0, 1);
435. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);
436. glRasterPos4i(0, -1, 0, 1);
437. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);
438. **glFlush();**
439. }
440. static void key\_pressed(unsigned char key, int x, int y) {
441. switch(key){
442. **case 27:**
443. tidy\_and\_exit();
444. break;
445. default:
446. printf("**\n**Press escape to exit**\n**");**break;**
447. }
448. }
449. int time\_difference(struct timespec \*start, struct timespec \*finish,
     * + 1. **long long int \*difference) {**
450. long long int ds = finish->tv\_sec - start->tv\_sec;
451. long long int dn = finish->tv\_nsec - start->tv\_nsec;
452. if(dn < 0 ) {
453. **ds--;**
454. dn += 1000000000;
455. }
456. \*difference = ds \* 1000000000 + dn;
457. return !(\*difference > 0);
458. **}**
459. int main(int argc, char \*\*argv) {
460. unsigned char \*d\_results;
461. **unsigned char \*d\_image;**
462. cudaMalloc((void\*\*)&d\_image, sizeof(unsigned char) \* (width \* height));
463. cudaMalloc((void\*\*)&d\_results, sizeof(unsigned char) \* (width \* height));
464. cudaMemcpy(d\_image, &image, sizeof(unsigned char) \* (width \* height), cudaMemcpyHostToDevice);
465. signal(SIGINT, sigint\_callback);
466. **struct timespec start, finish;**
467. long long int time\_elapsed;
468. clock\_gettime(CLOCK\_MONOTONIC, &start);
469. detect\_edges<<<100,72>>>(d\_image, d\_results);
470. **cudaThreadSynchronize();**
471. cudaMemcpy(&results, d\_results, sizeof(unsigned char) \* (width \* height), cudaMemcpyDeviceToHost);
472. clock\_gettime(CLOCK\_MONOTONIC, &finish);
473. time\_difference(&start, &finish, &time\_elapsed);
474. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed,
     * 1. (time\_elapsed/1.0e9));
475. **cudaFree(&d\_image);**
476. cudaFree(&d\_results);
477. glutInit(&argc, argv);
478. glutInitWindowSize(width \* 2,height);
479. **glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);**
480. glutCreateWindow("6CS005 Image Progessing Courework");
481. glutDisplayFunc(display);
482. glutKeyboardFunc(key\_pressed);
483. **glClearColor(0.0, 1.0, 0.0, 1.0);**
484. glutMainLoop();
485. tidy\_and\_exit();
486. return 0;
487. }

**2.2.1 Explanation of CUDA Image Processing:**

CUDA is known for parallel computing which uses graphical processing unit. The run time for the program decreases substantially if the process is done in GPU instead of CPU. The program flow is explained below:

To calculate time two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. Time elapsed is calculated by subtracting finish time and start time after getting the both the time. The CUDA Image processing is done in sequential manner. Following steps take place:

1. Declaration and allocation of both host memory (CPU) and device memory (GPU)

2. the host data initialized.

3. the data was transferred to device (GPU) from the host (CPU).

4. The host execute all the program

5. The result was transferred to the host from device.

Two positive variables \*d\_results and \*d\_image with datatype char are declared. CudaMalloc() function is used to allocate linear memory in the Grapical processing unit. CudaMalloc() function accepts two parameter. One is pointer to allocated device memory and another is request allocation size in bytes. In our program we have to allocate memory for two of the variables that is &d\_results and &d\_image. CudaMemcpy() function with parameter cudaMemcpyHostToDevice helps to copies the buffers from host to device memory for processing and cudaMemcpyDeviceToHost

copying the result back to host computer from device memory. First, we have to send the image and area to the device with cudaMemcpyHostToDevice parameter. Then after that detect\_edges <<100,72>> function which will execute in 100 thread blocks consisting of 72 thread. Triple Angular bracket is also known as kernel launch. The kernel function will be called up only one time and \_\_global\_\_ function is used which will execute on GPU and called from CPU. Inside \_\_global\_\_ function named detect\_edges() which requires the two-parameter image and result. Image is denoted by in pointer and result is denoted by out pointer. Since each block handles different array elements, each array is indexed using blockIdx.x. Similarly, threadIdx.x is used to access the thread index. blockIdx.x is multipled by the height of the picture and threadIdx.x is added to that.

Our program uses convolution formula to detect the edge of the given picture. To detect the edge x, y pixels are taken from the image and defined as the

y = i / width;

x = i - (width \* y);

The value of result can be calculated by using the value obtaining formula given by convolution theorem:

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

If the value of r is greater than 0 that is positive, edge of output result is set to white color(255) and if value is not equal to zero then the output result will be set to black color.Before calculating the time taken by the program cudaThreadSynchronize() is called to block until all the previous requested tasks have been completed by the system. After all of this finish time is taken to calculate the total time taken by the program with the help of time\_difference() function.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (CUDA) |
| 1 | 65933ns or 0.000065933s | 185296ns or 0.000185296s |
| 2 | 115084ns or 0.000115084s | 201635ns or 0.000201635s |
| 3 | 64197ns or 0.000064197s | 233052ns or 0.000233052s |
| 4 | 65573ns or 0.000065573s | 179347ns or 0.000179347s |
| 5 | 66473ns or 0.000066473s | 223546ns or 0.000223546s |
| 6 | 122975ns or 0.000122975s | 213171ns or 0.000213171s |
| 7 | 94871ns or 0.000094871s | 221334ns or 0.000221334s |
| 8 | 68225ns or 0.000068225s | 182270ns or 0.000182270s |
| 9 | 63991ns or 0.000063991s | 307762ns or 0.000307762s |
| 10 | 63396ns or 0.000063396s | 205284ns or 0.000205284s |
|  | Mean Running time for original program: 79071.8ns or 0.0000790718s | Mean Running time for CUDA program: 215269.7ns or 0.0002152697s |

Table 7 Comparison between original and CUDA version of image processing

**2.2.3 Write a short analysis of the results:**

CUDA is developed by Nvidia to improve the speed and pace by using the Graphical Processing unit. The mean running time for original program is 79071.8ns or 0.0000790718s. We have already used multithreading posix which gives the running time of 2970328.7ns or 0.0029703287s. After applying CUDA in image processing to detect the edge of the image, it was found to be 215269.7ns or 0.0002152697s which is very more than posix and original program after comparing the result, it is found that the CUDA program is very slower than both the program. It is due to the fact that the each cuda threads process each pixel. The number of pixels of the image is calculated by multiplying width and height of image which result 7200 pixels. So, 7200 thread process takes longer time than original as CUDA version tries to solve parallelly. Moreover, CUDA program takes couple of second to copy and transfer data from GPU to CPU where as in original and multithread, the programs are processed in CPU.

## 2.3 Linear Regression

**Code for CUDA Linear Regression:**

1. #include <stdio.h>
2. #include <math.h>
3. #include <time.h>
4. #include <unistd.h>
5. **#include <cuda\_runtime\_api.h>**
6. #include <errno.h> 7. #include <unistd.h>

8.

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. ***\* The variable names and the function names of this program is same as provided by the university.***
3. *The added variable and function are the only changes made to this program.*
4. *\* To compile:*
5. ***\* nvcc -o bishallinear bishallinear.cu -lm***
6. *\**
7. *\* To run:*
8. *\* ./bishallinear*
9. *\**
10. ***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

21.

1. typedef struct point\_t {
2. double x;
3. double y;
4. **} point\_t;**

26.

1. int n\_data = 1000;
2. \_\_device\_\_ int d\_n\_data = 1000;

29.

**30.**

1. point\_t data[] = {
2. {72.12,100.78},{65.40,107.86},{82.27,131.60},{82.31,122.34},
3. {89.41,121.50},{71.37,113.51},{82.62,112.38},{69.57,102.96},
4. {65.38,99.27},{84.50,138.85},{87.18,114.17},{73.03,109.21},
5. {67.26,102.06},{72.25,113.23},{61.28,101.59},{41.60,84.24},
6. {40.14,57.03},{15.24,45.58},{61.88,89.90},{34.89,72.77},
7. { 8.91,36.34},{30.45,46.18},{67.93,89.35},{68.82,112.80},
8. {63.96,99.32},{32.36,56.12},{42.20,63.66},{24.47,60.75},
9. { 1.96,28.62},{41.42,68.41},{34.49,73.14},{ 8.03,22.13},
10. {80.55,117.79},{85.54,130.80},{68.99,103.13},{99.32,144.79},
11. {91.71,153.61},{71.17,108.40},{85.28,120.11},{99.52,128.68},
12. {13.24,31.67},{ 5.19,40.15},{ 9.84,57.36},{29.42,54.01},
13. {89.68,126.25},{29.45,41.30},{79.63,132.59},{71.88,107.31},
14. {20.05,48.38},{40.98,54.11},{56.55,63.61},{77.22,114.17},
15. {63.86,88.10},{92.93,134.84},{56.84,101.20},{34.31,71.18},
16. {93.89,116.43},{38.02,63.78},{61.25,94.71},{71.02,103.42},
17. {95.05,142.82},{96.24,133.50},{19.50,50.92},{41.14,70.59},
18. {91.49,134.05},{54.05,98.31},{36.59,68.48},{91.14,130.45},
19. {44.76,88.98},{77.28,138.16},{64.80,96.33},{43.25,70.08},
20. {55.55,95.70},{ 3.77,39.03},{ 3.23,44.69},{86.72,127.42},
21. {84.62,131.54},{26.13,71.24},{61.22,98.22},{53.90,96.07},
22. {64.81,109.35},{91.66,116.79},{53.65,104.81},{38.42,66.16},
23. {62.33,112.41},{ 7.41,29.86},{41.59,57.59},{56.49,91.60},
24. {15.94,42.82},{97.46,140.29},{57.17,85.11},{26.94,45.86},
25. {73.14,96.37},{18.61,60.58},{15.69,44.16},{20.79,33.86},
26. {65.02,106.03},{38.09,72.71},{87.15,116.68},{77.45,123.08},
27. {90.47,126.33},{26.80,44.96},{75.94,119.76},{33.83,69.11},
28. {63.59,103.98},{38.05,72.36},{68.28,110.76},{ 3.34,54.22},
29. {45.40,92.84},{78.37,113.49},{27.11,46.46},{32.32,68.44},
30. {20.97,30.90},{37.92,75.11},{96.85,130.96},{69.40,95.17},
31. { 3.29,30.06},{64.41,103.44},{15.80,52.64},{61.76,97.79},
32. { 1.62,33.98},{29.03,58.02},{18.74,34.93},{25.41,73.73},
33. {28.78,65.94},{14.64,50.31},{82.85,133.70},{41.62,90.32},
34. {99.28,144.95},{90.16,133.18},{40.45,77.72},{ 1.79,50.44},
35. {31.80,62.71},{26.30,40.89},{47.57,83.15},{17.78,44.90},
36. {69.48,93.13},{87.98,126.95},{69.84,106.00},{37.06,61.61},
37. {90.65,133.97},{10.73,46.60},{38.84,79.90},{ 4.75,33.89},
38. {48.99,89.31},{ 2.51,47.09},{34.99,86.40},{29.79,54.52},
39. {91.30,133.72},{74.12,122.86},{90.93,141.88},{51.14,89.93},
40. {84.53,142.49},{26.84,58.79},{ 6.95,20.98},{49.80,85.14},
41. {22.82,57.02},{44.08,89.32},{22.28,48.72},{21.12,50.68},
42. {65.69,93.93},{27.84,39.97},{ 1.92,40.39},{ 9.36,33.54},
43. {88.10,123.02},{18.15,63.84},{21.80,39.76},{64.42,101.03},
44. { 2.23,22.52},{55.68,99.56},{37.55,87.77},{74.23,104.87},
45. {11.96,37.30},{23.60,45.84},{11.13,34.32},{ 9.05,48.79},
46. {56.11,100.21},{19.31,54.44},{ 6.27,16.17},{64.65,101.39},
47. {50.25,77.59},{69.33,95.12},{47.52,87.79},{28.97,65.98},
48. {71.56,95.30},{19.71,41.47},{57.66,96.65},{41.07,74.10},
49. {35.08,79.46},{40.80,87.01},{ 0.31,19.82},{90.78,111.55},
50. {34.39,72.03},{99.97,139.40},{30.86,73.03},{14.37,50.15},
51. { 6.11,42.76},{21.75,80.30},{89.94,127.56},{10.86,42.40},
52. {13.07,42.98},{84.47,147.14},{83.44,132.18},{32.24,63.57},
53. {66.93,102.41},{34.48,68.96},{ 3.46,22.82},{94.84,130.83},
54. {49.41,107.26},{71.64,99.82},{47.28,80.62},{39.17,68.77},
55. {58.05,108.35},{69.27,109.81},{47.64,73.34},{34.64,73.15},
56. {22.86,46.34},{37.76,66.19},{ 3.12,39.11},{60.59,111.05},
57. {91.99,122.76},{96.60,138.86},{ 3.58,23.35},{22.81,60.18},
58. {13.93,21.32},{69.51,106.41},{19.57,43.39},{79.11,115.68},
59. {80.89,124.36},{44.42,57.78},{33.28,73.04},{21.45,49.88},
60. {70.57,113.77},{45.63,65.60},{55.99,72.21},{21.62,41.47},
61. {61.74,98.99},{ 9.30,29.77},{75.32,106.74},{27.97,73.44},
62. {74.77,115.98},{42.93,82.67},{92.32,138.05},{25.55,64.34},
63. { 0.48,23.51},{79.52,111.52},{52.83,70.58},{51.45,87.28},
64. {62.72,90.41},{ 4.16,40.60},{70.13,115.25},{55.96,97.34},
65. {93.88,154.09},{46.21,90.04},{34.75,51.46},{54.45,89.56},
66. {80.69,129.36},{45.14,73.00},{47.34,85.69},{70.16,118.02},
67. { 4.26,17.14},{61.56,98.04},{15.95,28.56},{74.06,118.48},
68. {65.29,99.71},{19.08,55.64},{37.82,72.36},{58.22,103.93},
69. {50.52,82.15},{26.25,60.91},{97.77,123.91},{39.13,68.03},
70. {15.09,41.88},{32.61,61.64},{11.23,22.85},{61.92,98.02},
71. {73.63,126.32},{35.12,54.74},{12.98,42.69},{83.87,128.60},
72. {45.65,78.81},{42.85,90.57},{76.74,117.53},{19.05,49.60},
73. {69.03,104.16},{23.66,54.97},{52.85,85.94},{82.07,128.27},
74. {74.77,111.22},{95.04,136.69},{40.49,49.53},{ 4.16,28.40},
75. { 7.69,51.29},{29.37,80.82},{86.06,122.19},{ 3.92,23.24},
76. {62.76,108.89},{27.12,54.24},{10.24,33.84},{79.86,107.97},
77. {57.09,85.27},{10.29,54.38},{53.50,82.98},{12.83,50.29},
78. { 2.09,13.69},{88.73,135.16},{42.72,87.10},{40.20,91.88},
79. {40.10,76.49},{80.22,133.65},{57.55,93.99},{29.34,69.08},
80. { 2.90,41.26},{44.60,82.03},{47.93,89.05},{98.17,123.11},
81. {17.21,45.91},{42.37,79.83},{90.89,119.42},{ 7.81,36.64},
82. {76.14,123.86},{47.79,83.40},{95.27,144.30},{44.13,98.20},
83. {19.97,37.36},{90.66,131.96},{75.41,117.80},{57.14,107.91},
84. {25.92,41.69},{90.86,130.36},{44.78,79.02},{23.00,29.10},
85. {91.67,118.13},{26.55,51.18},{41.60,74.91},{ 0.39, 6.79},
86. {86.31,102.08},{20.43,37.80},{ 5.39,28.65},{12.63,24.33},
87. {22.60,42.79},{ 1.77,14.54},{74.10,113.64},{54.46,87.67},
88. {18.64,49.32},{93.97,116.30},{42.62,87.04},{13.37,30.16},
89. {74.50,104.62},{18.28,67.85},{76.98,107.84},{25.89,57.35},
90. {13.52,42.87},{61.26,97.78},{ 5.97,31.34},{91.99,137.43},
91. {20.38,58.23},{ 9.59,31.56},{79.41,126.40},{89.90,134.36},
92. {73.18,111.44},{61.51,111.41},{99.96,147.82},{72.55,113.52},
93. {66.21,110.93},{36.47,59.41},{65.58,93.39},{24.93,51.71},
94. {58.00,95.89},{49.83,83.52},{53.35,89.98},{83.97,129.85},
95. {57.33,106.86},{53.94,98.13},{98.02,144.26},{47.28,72.52},
96. {45.48,100.70},{80.69,147.66},{96.14,140.01},{82.69,120.80},
97. {79.73,136.89},{11.42,27.51},{88.91,138.59},{25.53,51.26},
98. { 2.49,37.14},{63.89,93.28},{90.96,138.02},{15.27,53.03},
99. {25.39,51.31},{31.77,55.54},{88.25,124.46},{67.66,108.26},
100. {90.23,112.02},{17.40,43.85},{78.38,137.07},{96.28,149.45},
101. {77.38,120.54},{56.49,107.27},{99.00,141.67},{36.35,58.18},
102. {97.41,132.64},{15.03,48.28},{42.48,81.20},{62.95,105.32},
103. {99.76,147.11},{85.18,140.95},{99.23,131.84},{21.09,44.44},
104. {45.12,75.22},{80.36,119.71},{61.37,84.74},{82.64,128.58},
105. {70.34,108.16},{83.63,116.26},{47.73,67.57},{17.56,48.42},
106. {23.26,42.12},{41.81,82.17},{18.48,33.63},{39.11,70.14},
107. {84.20,123.97},{67.20,113.97},{52.74,87.79},{81.66,131.54},
108. {45.90,93.69},{20.82,34.77},{86.35,122.38},{78.93,106.82},
109. {10.56,44.66},{51.20,104.61},{93.79,131.97},{15.71,43.06},
110. {99.16,156.47},{90.70,135.27},{41.85,77.91},{73.41,106.66},
111. {57.51,108.55},{53.06,115.27},{25.72,67.45},{ 8.03,27.74},
112. {57.91,101.56},{35.87,57.47},{98.33,145.81},{50.96,76.84},
113. {57.86,102.10},{17.21,44.21},{95.62,154.59},{76.92,114.77},
114. {25.32,60.66},{43.60,68.34},{42.68,73.98},{60.36,84.81},
115. { 9.06,42.91},{ 4.16,18.44},{54.14,97.87},{ 4.87,35.92},
116. {75.38,112.62},{41.37,68.92},{88.16,163.96},{16.79,41.87},
117. { 9.77,40.62},{69.66,125.12},{70.35,118.66},{71.99,97.87},
118. {63.66,111.29},{ 2.01,19.46},{64.63,122.89},{48.39,84.19},
119. {28.15,64.69},{46.17,83.91},{25.12,45.94},{82.23,118.70},
120. {57.69,95.98},{24.42,62.91},{15.81,35.58},{75.28,106.87},
121. {95.74,133.25},{67.78,107.42},{80.89,128.72},{10.39,38.37},
122. {15.31,35.73},{61.45,110.46},{11.15,44.99},{30.80,63.26},
123. {84.29,122.39},{29.17,47.34},{80.68,138.44},{81.17,117.86},
124. { 8.47,32.78},{41.26,74.09},{43.50,71.18},{34.48,68.61},
125. {30.63,68.05},{88.63,137.28},{71.56,116.97},{21.03,39.12},
126. {88.20,116.24},{ 8.52,30.24},{95.79,137.27},{78.66,104.62},
127. {72.44,94.21},{71.60,106.34},{72.11,114.18},{34.50,59.18},
128. {22.85,60.95},{18.43,40.91},{69.24,119.69},{91.84,142.06},
129. {34.41,69.95},{95.06,136.92},{67.93,100.93},{46.96,71.82},
130. {63.92,102.14},{ 1.62,29.66},{95.24,133.60},{43.10,80.88},
131. {21.83,73.25},{35.01,62.42},{20.05,55.19},{18.64,45.92},
132. {40.28,75.26},{34.54,63.38},{84.74,117.68},{90.38,144.87},
133. { 9.91,24.87},{62.97,102.14},{34.40,79.20},{67.34,89.48},
134. {48.53,85.13},{24.57,51.59},{81.95,117.78},{22.23,49.77},
135. {75.86,125.20},{60.45,99.78},{19.93,35.57},{48.62,78.46},
136. {88.49,120.71},{13.33,40.67},{52.03,93.38},{38.43,80.28},
137. { 2.56,17.00},{18.39,58.10},{58.81,88.08},{75.76,96.69},
138. {69.78,98.83},{96.47,146.81},{47.32,79.89},{21.90,46.54},
139. {52.39,83.38},{75.49,107.96},{50.14,80.51},{41.54,73.80},
140. {76.07,117.48},{27.00,73.59},{81.59,122.88},{21.74,39.55},
141. {60.05,105.04},{75.68,102.72},{40.41,79.01},{ 0.32,24.82},
142. {50.06,106.14},{98.69,139.50},{64.17,109.26},{42.74,78.53},
143. {39.52,71.78},{55.14,97.37},{25.19,39.08},{99.31,142.63},
144. {67.50,91.86},{90.92,152.17},{81.99,129.38},{77.28,124.08},
145. {29.38,69.15},{ 3.81,41.93},{ 9.72,41.83},{25.75,53.09},
146. {57.28,85.11},{69.50,116.90},{20.00,51.46},{63.00,72.32},
147. {67.06,102.20},{37.85,64.86},{81.40,114.28},{13.32,58.41},
148. {67.21,103.77},{63.73,109.66},{91.43,141.66},{54.83,88.07},
149. {68.03,112.67},{ 0.51,27.76},{ 2.17,38.05},{36.26,66.58},
150. {72.67,116.52},{98.28,136.37},{85.27,128.64},{90.26,136.47},
151. {60.31,95.24},{32.77,58.94},{ 3.52,24.75},{15.98,45.49},
152. {94.25,145.90},{ 8.13,29.89},{61.13,81.38},{44.14,77.64},
153. {63.53,100.35},{49.35,97.92},{ 4.98,32.12},{25.53,57.45},
154. { 8.63,41.62},{24.23,56.27},{93.30,137.92},{43.72,71.72},
155. {54.15,89.12},{ 3.42,36.34},{57.75,85.68},{51.90,87.74},
156. {85.14,137.82},{99.27,173.87},{82.53,124.94},{15.38,44.42},
157. {66.66,108.56},{64.12,99.41},{39.08,73.77},{25.42,58.25},
158. { 1.29,36.39},{98.72,148.84},{70.09,112.06},{ 8.51,27.00},
159. {85.92,124.74},{88.32,127.04},{51.79,74.58},{36.46,62.45},
160. {49.29,85.33},{14.06,30.58},{24.83,34.82},{42.85,87.06},
161. {34.47,76.96},{59.16,90.44},{ 1.02,32.32},{61.80,108.22},
162. {72.52,95.83},{65.40,99.49},{53.32,93.79},{74.22,117.61},
163. {53.86,88.31},{39.84,80.11},{79.28,117.86},{34.57,76.73},
164. {21.69,55.55},{99.87,129.34},{72.12,108.86},{75.08,106.64},
165. {70.71,106.00},{18.35,67.45},{37.42,66.71},{ 0.70, 9.02},
166. {56.79,86.75},{74.04,100.45},{53.40,82.23},{42.13,70.45},
167. {82.43,123.55},{91.65,131.55},{94.99,153.70},{62.14,84.17},
168. {99.71,151.07},{33.24,73.77},{48.87,76.91},{68.57,118.95},
169. {14.28,46.22},{18.17,41.01},{95.93,133.32},{ 5.06,33.23},
170. {57.58,95.47},{18.71,39.10},{90.19,136.73},{26.98,50.08},
171. {11.36,26.14},{62.70,98.59},{49.32,80.54},{99.97,149.27},
172. {83.40,132.00},{25.30,48.62},{79.25,117.83},{81.09,109.23},
173. {31.46,51.02},{14.26,32.26},{33.53,52.63},{ 9.42,47.16},
174. {67.40,109.90},{18.56,32.79},{34.51,75.14},{49.00,77.38},
175. {15.69,50.80},{23.09,40.32},{32.03,67.86},{13.60,40.35},
176. {19.21,60.16},{78.56,111.57},{80.72,131.02},{50.19,79.64},
177. {55.60,81.78},{ 6.37,43.37},{42.78,74.85},{60.48,113.67},
178. {44.44,89.27},{54.02,90.24},{73.51,101.74},{16.41,56.73},
179. {70.94,104.90},{32.03,66.91},{13.12,49.71},{50.16,85.64},
180. {41.31,68.88},{69.25,123.25},{24.97,69.28},{40.80,86.30},
181. {32.28,67.01},{90.77,142.80},{66.77,104.70},{24.06,56.12},
182. {49.16,89.52},{46.10,95.56},{51.79,94.01},{56.11,100.66},
183. {88.49,126.71},{ 1.28,21.35},{35.55,64.10},{18.79,29.74},
184. { 5.40,40.02},{92.32,129.89},{21.13,47.05},{ 5.14,32.16},
185. {60.89,104.41},{43.45,76.07},{98.91,160.53},{99.31,155.80},
186. {74.71,121.53},{62.33,98.98},{58.66,101.10},{51.51,93.03},
187. {51.69,90.42},{19.47,31.22},{85.75,108.87},{64.20,100.48},
188. {96.60,142.66},{67.99,102.48},{68.37,120.07},{29.81,44.77},
189. {96.55,142.74},{30.59,43.25},{73.94,108.44},{49.77,88.88},
190. {59.48,98.21},{41.21,61.86},{38.63,83.41},{86.98,140.40},
191. {93.34,134.69},{87.92,119.52},{40.93,61.87},{ 2.43,30.68},
192. {50.74,71.81},{37.13,52.43},{ 1.50,22.18},{99.06,143.48},
193. { 1.67,27.67},{ 0.18,10.50},{54.13,77.05},{46.19,88.91},
194. {91.13,144.49},{ 8.95,28.33},{85.69,122.61},{50.30,95.60},
195. {48.63,103.49},{67.99,100.19},{69.21,112.13},{11.26,34.99},
196. {25.78,58.73},{84.35,112.36},{46.80,79.68},{69.54,117.99},
197. {40.30,74.33},{79.97,118.95},{23.28,55.71},{32.62,78.92},
198. {21.86,37.01},{ 5.07,22.57},{94.41,146.15},{40.14,60.81},
199. {95.80,125.35},{91.34,131.68},{72.55,113.56},{40.13,71.59},
200. {98.06,145.27},{90.55,144.08},{71.26,121.81},{33.85,71.13},
201. {85.74,142.63},{57.93,91.78},{ 7.63,39.30},{83.72,128.26},
202. {10.89,46.78},{39.79,66.98},{98.84,146.32},{84.62,123.91},
203. {23.16,31.94},{86.36,134.79},{44.19,63.74},{ 0.39,24.19},
204. {64.22,96.97},{66.47,103.78},{ 1.73,17.52},{22.25,36.77},
205. {31.88,59.39},{15.60,30.03},{16.08,41.91},{83.11,129.19},
206. {72.61,122.52},{19.02,41.06},{56.90,87.53},{65.85,97.02},
207. {81.40,120.35},{64.90,104.44},{73.35,119.00},{ 8.49,40.31},
208. {31.20,65.32},{28.29,75.05},{72.51,120.90},{20.42,48.84},
209. {71.46,111.59},{33.98,50.46},{72.48,111.29},{75.56,113.00},
210. {58.65,95.16},{23.66,44.95},{95.08,139.46},{80.12,115.20},
211. {67.77,101.97},{56.06,99.08},{99.03,138.47},{48.26,74.79},
212. {25.95,39.30},{85.20,137.70},{69.31,104.19},{86.19,122.91},
213. {37.99,87.47},{72.06,116.90},{ 5.66,28.92},{27.77,52.05},
214. {31.89,60.32},{18.01,48.92},{37.21,65.49},{73.76,107.20},
215. { 0.32,-0.71},{93.75,133.48},{69.11,109.63},{11.01,55.84},
216. {43.48,73.99},{20.76,57.44},{75.50,105.00},{98.74,150.46},
217. {40.75,90.93},{61.67,103.30},{93.48,155.96},{35.52,61.62},
218. {32.30,78.52},{28.92,49.61},{60.97,87.11},{13.59,47.58},
219. { 9.43,26.07},{58.00,107.90},{99.86,151.90},{34.01,57.82},
220. {39.02,59.14},{33.64,74.99},{ 2.28,20.21},{55.00,90.93},
221. {55.77,85.94},{79.17,134.03},{63.16,106.70},{17.58,32.28},
222. {24.29,34.68},{83.91,132.35},{96.44,129.86},{61.95,93.66},
223. {14.86,25.10},{15.53,33.29},{15.69,42.47},{80.60,126.11},
224. {16.01,46.33},{26.54,74.55},{ 2.67,37.10},{74.63,96.98},
225. {38.06,59.99},{56.59,96.87},{78.88,120.95},{87.56,121.75},
226. {73.54,119.27},{16.84,44.09},{44.24,89.36},{76.02,123.64},
227. {98.41,115.45},{12.11,48.19},{30.70,60.41},{55.51,100.49},
228. { 0.26,37.11},{83.43,124.44},{49.92,111.30},{65.55,99.48},
229. {77.61,119.44},{62.44,95.52},{21.80,61.06},{20.99,60.54},
230. {93.10,129.45},{54.96,91.05},{10.22,48.48},{66.77,108.83},
231. {40.83,87.14},{13.54,35.77},{31.44,62.92},{79.69,110.30},
232. {67.07,100.59},{28.81,78.71},{52.95,97.30},{39.89,81.67},
233. {58.79,75.89},{34.35,51.29},{38.03,64.97},{87.87,130.19},
234. {39.73,52.43},{ 1.64,31.22},{91.15,147.58},{54.08,101.10},
235. {53.53,74.54},{54.24,104.47},{15.04,51.28},{79.06,114.59},
236. {93.83,138.37},{94.89,122.18},{52.63,86.22},{27.83,68.05},
237. {54.51,94.07},{23.83,58.00},{86.88,141.66},{10.42,31.81},
238. {55.43,84.31},{45.04,85.30},{95.69,121.78},{17.28,35.32},
239. { 3.17,33.76},{51.61,69.81},{27.37,64.13},{88.92,160.98},
240. {31.40,64.46},{33.35,59.91},{82.48,128.89},{50.46,98.13},
241. {78.73,113.68},{70.08,115.27},{98.65,142.28},{ 9.15,50.95},
242. {16.74,35.73},{32.92,72.02},{ 1.29,18.94},{75.79,123.45},
243. {32.94,59.92},{61.72,81.50},{42.39,91.90},{70.15,108.81},
244. { 2.90,29.10},{59.68,87.41},{69.85,108.66},{71.21,107.81},
245. {24.09,46.47},{44.51,76.59},{ 7.30,34.83},{58.93,99.24},
246. { 1.24,22.60},{84.27,132.21},{54.11,87.19},{39.18,75.93},
247. {90.81,155.72},{67.68,88.19},{67.14,84.53},{53.98,86.47},
248. {67.28,106.68},{ 8.49,36.74},{34.96,62.55},{59.01,82.94},
249. {64.78,101.77},{66.24,110.82},{75.81,131.28},{62.82,76.02},
250. {73.95,116.37},{20.40,38.76},{45.06,84.65},{47.64,82.81},
251. {30.85,64.41},{77.10,112.67},{ 8.12,32.76},{39.56,53.41}
252. };

284. double residual\_error(double x, double y, double m, double c) {

**285. double e = (m \* x) + c - y;**

1. return e \* e;
2. } 288.
3. \_\_device\_\_ double d\_residual\_error(double x, double y, double m, double c) {
4. **double e = (m \* x) + c - y;**
5. return e \* e;
6. } 293.
7. double rms\_error(double m, double c) {
8. **int i;**
9. double mean; 297. double error\_sum = 0;

298.

1. for(i=0; i<n\_data; i++) {
2. **error\_sum += residual\_error(data[i].x, data[i].y, m, c);**
3. }

303. mean = error\_sum / n\_data;

304.

1. **return sqrt(mean);**
2. } 307.

308. \_\_global\_\_ void d\_rms\_error(double \*m, double \*c, double \*error\_sum\_arr, point\_t \*d\_data) {

309.

**310. int i = threadIdx.x + blockIdx.x \* blockDim.x;**

311.

1. error\_sum\_arr[i] = d\_residual\_error(d\_data[i].x, d\_data[i].y, \*m, \*c);
2. }
3. **int time\_difference(struct timespec \*start, struct timespec \*finish,**
4. long long int \*difference) {
5. long long int ds = finish->tv\_sec - start->tv\_sec;
6. long long int dn = finish->tv\_nsec - start->tv\_nsec;

319.

1. **if(dn < 0 ) {**
2. ds--;
3. dn += 1000000000;
4. }
5. \*difference = ds \* 1000000000 + dn;
6. **return !(\*difference > 0);**
7. } 327.
8. int main() {
9. int i;
10. **double bm = 1.3;**
11. double bc = 10;
12. double be;
13. double dm[8];
14. double dc[8];
15. **double e[8];**
16. double step = 0.01;
17. double best\_error = 999999999;
18. int best\_error\_i; 339. int minimum\_found = 0;

**340.**

341. double om[] = {0,1,1, 1, 0,-1,-1,-1}; 342. double oc[] = {1,1,0,-1,-1,-1, 0, 1};

343.

1. struct timespec start, finish;
2. **long long int time\_elapsed;**

346.

347.

1. clock\_gettime(CLOCK\_MONOTONIC, &start);
2. **cudaError\_t error;**

351.

352.

1. double \*d\_dm;
2. double \*d\_dc;
3. **double \*d\_error\_sum\_arr;**
4. point\_t \*d\_data; 357.

358. be = rms\_error(bm, bc);

359.

**360.**

1. error = cudaMalloc(&d\_dm, (sizeof(double) \* 8));
2. if(error){
3. fprintf(stderr, "cudaMalloc on d\_dm returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. **exit(1);**
6. } 367.

368.

1. error = cudaMalloc(&d\_dc, (sizeof(double) \* 8));
2. **if(error){**
3. fprintf(stderr, "cudaMalloc on d\_dc returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. exit(1);
6. }

1. error = cudaMalloc(&d\_error\_sum\_arr, (sizeof(double) \* 1000));
2. if(error){
3. fprintf(stderr, "cudaMalloc on d\_error\_sum\_arr returned %d %s**\n**", error,
4. **cudaGetErrorString(error));**
5. exit(1);
6. } 383.

384.

1. **error = cudaMalloc(&d\_data, sizeof(data));**
2. if(error){
3. fprintf(stderr, "cudaMalloc on d\_data returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. exit(1);
6. **}** 391.
7. while(!minimum\_found) {
8. for(i=0;i<8;i++) {
9. dm[i] = bm + (om[i] \* step);
10. **dc[i] = bc + (oc[i] \* step);**
11. } 397.

398.

1. error = cudaMemcpy(d\_dm, dm, (sizeof(double) \* 8), cudaMemcpyHostToDevice);
2. **if(error){**
3. fprintf(stderr, "cudaMemcpy to d\_dm returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. }

**405.**

1. error = cudaMemcpy(d\_dc, dc, (sizeof(double) \* 8), cudaMemcpyHostToDevice);
2. if(error){
3. fprintf(stderr, "cudaMemcpy to d\_dc returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. **}**

412.

1. error = cudaMemcpy(d\_data, data, sizeof(data), cudaMemcpyHostToDevice);
2. if(error){
3. **fprintf(stderr, "cudaMemcpy to d\_data returned %d %s\n", error,**
4. cudaGetErrorString(error));
5. }

419. for(i=0;i<8;i++) {

**420.**

1. double h\_error\_sum\_arr[1000];
2. double error\_sum\_total;
3. double error\_sum\_mean;
4. d\_rms\_error <<<100,10>>>(&d\_dm[i], &d\_dc[i], d\_error\_sum\_arr, d\_data);
5. **cudaThreadSynchronize();**
6. error = cudaMemcpy(&h\_error\_sum\_arr, d\_error\_sum\_arr, (sizeof(double) \* 1000), cudaMemcpyDeviceToHost);
7. if(error){
8. fprintf(stderr, "cudaMemcpy to error\_sum returned %d %s**\n**", error,
9. cudaGetErrorString(error));
10. **}**
11. for(int j=0; j<n\_data; j++) {
12. error\_sum\_total += h\_error\_sum\_arr[j];
13. }

434.

1. **error\_sum\_mean = error\_sum\_total / n\_data;**
2. e[i] = sqrt(error\_sum\_mean); 437.
3. if(e[i] < best\_error) {
4. best\_error = e[i];
5. **best\_error\_i = i;**
6. }
7. error\_sum\_total = 0;
8. }

446.

1. if(best\_error < be) {
2. be = best\_error;
3. bm = dm[best\_error\_i];  **bc = dc[best\_error\_i];**
4. } else {
5. minimum\_found = 1;
6. }
7. }

**455.**

1. error = cudaFree(d\_dm);
2. if(error){
3. fprintf(stderr, "cudaFree on d\_dm returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. **exit(1);**
6. }

462.

1. error = cudaFree(d\_dc);
2. if(error){
3. **fprintf(stderr, "cudaFree on d\_dc returned %d %s\n", error,**
4. cudaGetErrorString(error));
5. exit(1);
6. }

469.

1. **error = cudaFree(d\_data);**
2. if(error){
3. fprintf(stderr, "cudaFree on d\_data returned %d %s**\n**", error,
4. cudaGetErrorString(error));
5. exit(1);
6. **}**

476.

1. error = cudaFree(d\_error\_sum\_arr);
2. if(error){
3. fprintf(stderr, "cudaFree on d\_error\_sum\_arr returned %d %s**\n**", error,
4. **cudaGetErrorString(error));**
5. exit(1);
6. }

483.

484. printf("minimum m,c is %lf,%lf with error %lf**\n**", bm, bc, be);

**485.**

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

487.

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

489.

1. **printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,**
2. (time\_elapsed/1.0e9));

492.

1. return 0;
2. }

**Explanation of CUDA Linear Regression:**

The main motive behind this code is to find out the minimum m and c value with best error using CUDA toolkit. M and c are the slope or gradient of line and intercept of the straight line. Slope deals with direction and line’s steepness where intercepts are the point where it crosses the axis. There are intercepts x- intercept and y-intercept. We tried to find out the value of m and c with optimal root mean square error by following steps:

We have declared the bm, bc and best\_error with values 1.3, 10 and 999999999. We have also declared the best error in the program. Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively Time elapsed is calculated by subtracting finish time and start time after getting the both the time. After noting start time, error with datatyoe cudaError\_t is declared. After declaration phase, rms\_error() with parameter m and c is called. rms\_error() is the function to find the error of the straight line. Rms error can be calculated by square rooting all sum of squares of error divided by number of errors. Mathematically, it can be represented by:

Root Mean Square error=

Residual error is calculated by the difference between observed value and predicted value. Before finding the RMS error we must find the residual error to find the minimum error.

Residual error = observed value – predicted value

Inside rms\_error() function, another function residual\_error() function with parameter double x, double y, double m, double c. The value of x and y are taken from the data provided by the university to us. The residual error is calculated by the formula e = (m \* x) + c – y and square of e is returned to the function rms\_error(). CudaMalloc() function accepts two parameter. One is pointer to allocated device memory and another is request allocation size in bytes. In our program we have to allocate memory for 8 times the size of double because we are performing the program for 8 value of m and c. CudaMalloc() is used for vaues of m, c and error with &d\_dm, &d\_dc and &d\_error\_sum\_arr. If the program found error cudaGetErrorString() is printed in the terminal. CudaMemcpy() function with parameter cudaMemcpyHostToDevice helps to copies the buffers from host to device memory for processing.. cudaMemcpy is used for m, c and error in our program. and cudaMemcpyDeviceToHost helps to copy the result back to host computer from device memory.Then after that d\_rms\_error <<100,100>> function which will execute in 100 thread blocks consisting of 100 thread. Triple Angular bracket is also known as kernel launch. The kernel function will be called up only one time and \_\_global\_\_ function is used which will execute on GPU and called from CPU.

Inside \_\_global\_\_ function named rms\_error () which requires the four-parameters pointer. double \*m, double \*c, double \*error\_sum\_arr, and point\_t \*d\_data are four pointer for the value of m, c, error sum and data provided by the university.

Since each block handles different array elements, each array is indexed using blockIdx.x and threadIdx.x. blockDim.x is the number of block of thread at the specific dimension. Later for all the best error, the best error value is replaced when smaller value comes which helps us to find the minimum value of error. CudaThreadSynchronize() is called to block until all the previous requested tasks have been completed by the system. cudaFree() function is used to free up the spce used by the cudamalloc() function.So, cudafree() function is used for m, c and error pointer. CudaErrorInvalidDevicePointer is returned if the cuda\_free() function fails to free up the space. Then only the minimum value of m, c and error is displayed in the terminal by using print statement. The total time taken by the program is calculated by the function named time\_difference() function and displayed in the terminal.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (CUDA) |
| 1 | 52689676ns or 0.052689676s | 338443849ns or 0. 338443849s |
| 2 | 52694668ns or 0.052694668s | 329248076ns or 0. 329248076s |
| 3 | 52739608ns or 0.052739608s | 322433698ns or 0. 322433698s |
| 4 | 52767176ns or 0.052767176s | 326285789ns or 0. 326285789s |
| 5 | 52658329ns or 0.052658329s | 322764838ns or 0. 322764838s |
| 6 | 52725256ns or 0.052725256s | 325215315ns or 0. 325215315s |
| 7 | 53270258ns or 0.053270258s | 321968418ns or 0. 321968418s |
| 8 | 52675616ns or 0.052675616s | 321989362ns or 0. 321989362s |
| 9 | 53785147ns or 0.053785147s | 325985971ns or 0. 325985971s |
| 10 | 52968744ns or 0.052968744s | 323081038ns or 0. 323081038s |
|  | Mean Running time for original program: 52897447.8ns or 0.52897447s | Mean Running Time (CUDA version) 325741635.4ns or 0.3257416354s |

Table 8 Comparision between original and cuda verison of linear regression

**2.3.4 Write a short analysis of the results:**

As excepted, the mean running time for CUDA is lesser than the mean running time of original program. This is due to the fact that the CUDA program runs on GPU for better performance and CUDA is being optimised by Nvidia itself. We used 100 blocks with 100 threads per block. So, the program will work in 100 blocks and each block having 100 thread each. The mean Running time for original program is 52897447.8ns or 0.52897447s and the mean Running Time (CUDA version) 325741635.4ns or 0.3257416354s. The Cuda program is (0.52897447s- 0.3257416354s) = 0.203232835 seconds faster than the original program.

# 3. MPI

## 3.1 Password Cracking

**Code for MPI Password Cracking:**

2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <string.h>
5. **#include <time.h>**
6. #include <crypt.h>
7. #include <mpi.h>
9. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
10. ***The variable names and the function names of this program is same as provided by the university.***
11. *The added variable and function are the only changes made to this program.*

14. *To compile:*
15. ***mpicc -o MpiBishal BishalMpi.c -lrt -lcrypt***
17. *To run 3 processes on this computer:*
18. *mpirun -n 3 ./MpiBishal*
19. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

22. int n\_passwords = 4;
23. pthread\_t t1, t2;
24. char \*encrypted\_passwords[] = {
26. "$6$KB$H8s0k9/1RQ783G9gF69Xkn.MI.Dq5Ox0va/dFlkknNjO7trgekVOjTv1BKCb.nm3vqxmtO2mOplhmFkwZXecz0",
27. "$6$KB$VDUCASt5S88l82JzexhKDQLeUJ5zfxr16VhlVwNOs0YLiLYDciLDmN3QYAE80UIzfryYmpR.NFmbZvAGNoaHW.",
28. "$6$KB$LIQ21WOW2T678MQ//wddYygf4uuWaMWVM8vG9Pj9O.7sDQ1GVgmw8KqqNfjjU6BtBgJvE5nXbborTU/uwrnVz1",
29. "$6$KB$cU/6LrcqQGvsvRGllyfGDU1Wuvd1roXp.IEr3UYKQjBbT1sdP1PZPrZKrTGfFuRqD97EsgDalOwwn3OedIIjn0"
30. **};**
32. void substr(char \*dest, char \*src, int start, int length){
33. memcpy(dest, src + start, length);
34. \*(dest + length) = '**\0**';
35. **}**
37. void kernel\_function1(char \*salt\_and\_encrypted){
38. int x, y, z;
39. char salt[7];
41. char plain[7];
42. char \*enc;
43. int count = 0;
45. **substr(salt, salt\_and\_encrypted, 0, 6);**
47. for(x='A'; x<='M'; x++){
48. for(y='A'; y<='Z'; y++){
49. for(z=0; z<=99; z++){
50. **printf("Instance 1:");**
51. sprintf(plain, "%c%c%02d",x, y, z);
52. enc = (char \*) crypt(plain, salt);
53. count++;
54. if(strcmp(salt\_and\_encrypted, enc) == 0){
55. **printf("#%-8d%s %s\n", count, plain, enc);**
56. } else {
57. printf(" %-8d%s %s**\n**", count, plain, enc);
58. }
59. }
60. **}**
61. }
62. printf("%d solutions explored**\n**", count);
63. }
64. void kernel\_function2(char \*salt\_and\_encrypted){
65. **int x, y, z;**
66. char salt[7];
68. char plain[7];
69. char \*enc;
70. **int count = 0;**
72. substr(salt, salt\_and\_encrypted, 0, 6);
74. for(x='N'; x<='Z'; x++){
75. **for(y='A'; y<='Z'; y++){**
76. for(z=0; z<=99; z++){
77. printf("Instance 2:");
78. sprintf(plain, "%c%c%02d",x, y, z);
79. enc = (char \*) crypt(plain, salt);
80. **count++;**
81. if(strcmp(salt\_and\_encrypted, enc) == 0){
82. printf("#%-8d%s %s**\n**", count, plain, enc);
83. } else {
84. printf(" %-8d%s %s**\n**", count, plain, enc);
85. **}**
86. }
87. }
88. }
89. printf("%d solutions explored**\n**", count);
90. **}**

93. int time\_difference(struct timespec \*start, struct timespec \*finish,
94. long long int \*difference) {
95. **long long int ds = finish->tv\_sec - start->tv\_sec;**
96. long long int dn = finish->tv\_nsec - start->tv\_nsec;
98. if(dn < 0 ) {
99. ds--;
100. **dn += 1000000000;**
101. }
102. \*difference = ds \* 1000000000 + dn;
103. return !(\*difference > 0);
104. }
106. int main(int argc, char\*\* argv) {
107. struct timespec start, finish;
108. long long int time\_elapsed;
110. **clock\_gettime(CLOCK\_MONOTONIC, &start);**

113. int size, rank;
114. int i;
116. MPI\_Init(NULL, NULL);
117. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
118. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
119. if(size != 3) {
120. **if(rank == 0) {**
121. printf("This program needs to run on exactly 3 processes**\n**");
122. }
123. } else {
124. if(rank ==0){
126. int x;
128. MPI\_Send(&x, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);
129. MPI\_Send(&x, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);


133. } else if(rank==1){
134. int number;
135. **MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,**
136. MPI\_STATUS\_IGNORE);
137. for(i=0;i<n\_passwords;i<i++) {
138. kernel\_function1(encrypted\_passwords[i]);
139. }
140. **}**
141. else{
142. int number;
143. MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,
144. MPI\_STATUS\_IGNORE);
145. **for(i=0;i<n\_passwords;i<i++) {**
146. kernel\_function2(encrypted\_passwords[i]);
147. clock\_gettime(CLOCK\_MONOTONIC, &finish);
148. time\_difference(&start, &finish, &time\_elapsed);
149. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed,
150. **(time\_elapsed/1.0e9));**
152. }
153. }
154. }
155. **MPI\_Finalize();**
157. return 0;
158. }

**Explanation of MPI Password Cracking:**

MPI stands for Message passing Interface. MPI is portable and efficient used for parallel computing for high performance. The main aim of this code is to find out the runtime to crack the password using message passing interface. To calculate time two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameters CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. Time elapsed is calculated by subtracting finish time and start time after getting the both the time. In MPI, we must declare to variable size and time and MPI library is initialized by MPI\_Init datatype. MPI\_Comm\_size() function is used to identify size of communicator and MPI\_Comm\_rank() function is used to identify the rank of the calling process. The program only executes if there are exactly three process. The MPI program MPI\_Send() is used to do blocking send to other rank from the current rank. MPI\_Send has six parameters:

1. address of send buffer

2. count in send buffer

3. datatype of each send buffer element

4. rank of destination

5. message tag

6. communicator

We have used MPI\_Send() to send the work to the rank 1 and rank 2 from rank 0. We used MPI\_INT datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator.

Similarly, MPI\_Recv() is used to receive for sent message from another rank to the current rank and it has 7 parameters. They are:

1. address of send buffer

2. Count in send buffer

3. datatype of each send buffer element

4. Rank Source of the sender

5. message tag

6. communicator

7. Status

We have used MPI\_Recv() to receive the work from rank 0. We used MPI\_INT datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator. We ignored the status of the receive with MPI\_STATUS\_IGNORE keyword. We have sent our work from rank0 to rank1 and rank2. So, from rank1 and rank2 we have called the function kernel\_function1() and kernel\_function2() after receiving the message from rank 0. Kernel\_function1() tries to crack the passwords whose starting alphabet is A to M and Kernel\_function2() tries to crack the passwords whose starting alphabet from N to Z. Three loop variables x, y and z are used to find the matching password. We have used encryption along with six-digit ($6$KB$) salt in our password. So, in order to compare our loop encountered password with the provided password we must add salt with same six-digit. substr() function is used to separate first six digit from the salt and encrypted password and memcpy() function inside the substr() function with parameter destination, source and number is used to copy only the salt strings used in the encrypted password. And another function crypt() is used to encrypt the salt added plain password which later is compared to our password to find out the matching password. \*enc is the pointer to the encrypted password and count is the number of times the combination has explored. Strcmp() is another library function of c language which is used to compare the value of two variable. Here, we have used two variable salt\_and\_encrypted is the first string which has to be compared to another string named enc. In Strcmp() library function, it returns value zero if both strings are equal. So, if only the same value found, password found string is put before the count variable else other value like count, plain password and encrypted password is printed. Same procedure is applied to both the kernel functions to get the desired output. After completion, the total time taken by the program is calculated by the function named time\_difference() function. Since we have used MPI library specification, we must use MPI\_Finalize() function to terminates the execution environment of MPI.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (MPI) |
| 1 | 516.887633220s | 258597156294ns or 258.597156294s |
| 2 | 506.811146889s | 259512549865ns or 259.512549865s |
| 3 | 508.927593166s | 259336953257ns or 259.336953257s |
| 4 | 507.269538405s | 257759887638ns or 257.759887638s |
| 5 | 507.184282818s | 258514009585ns or 258.514009585s |
| 6 | 508.754546574s | 257278461172ns or 257.278461172s |
| 7 | 508.737758971s | 258865979999ns or 258.865979999s |
| 8 | 509.791337001s | 258328350516ns or 258.328350516s |
| 9 | 517.265658095s | 239451603712ns or 259.451603712s |
| 10 | 506.752829585s | 257210481794ns or 257.210481794s |
|  | Mean Running time for original program: 509838232472.4ns or 509.8382324724s | Mean Running time for MPI program:258.4855433832s |

Table 9 Comparison between original and MPI version password cracking

**Write a short analysis of the results:**

Using MPI library, the runtime can be reduced by parallel computing which help in better and high performance. After analysing the result, it found out that the MPI program runs in almost half the time of original program. The mean running time for original program is 509838232472.4ns or 509.8382324724s and the Mean Running time for MPI program is 258.4855433832s. So, the MPI program is (509.8382324724s / 258.4855433832s) = 1.97240520998 ≈ 2 times faster than the original program. This may be because of two processes is used to execute the program. The MPI program takes less time than

**Code for two initials four-digit password:**

2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <string.h>
5. **#include <time.h>**
6. #include <crypt.h>
7. #include <mpi.h>
9. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
10. ***The variable names and the function names of this program is same as provided by the university.***
11. *The added variable and function are the only changes made to this program.*

14. *To compile:*
15. ***mpicc -o mpibishal mpibishal.c -lrt -lcrypt***
17. *To run 3 processes on this computer:*
18. *mpirun -n 3 ./mpibishal > bihsalpwsixdigit.txt*
19. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

22. int n\_passwords = 4;
23. pthread\_t t1, t2;
24. char \*encrypted\_passwords[] = {
26. "$6$KB$JlnaB6F72rPwyQnDjKCreIFgzKEKcx365WM9/OH2gQjDdpxXoAYoXdPhnUzyGIvtzocmUX.mCzddTRd41Hxc7.",
28. "$6$KB$ptZuR.HTTY/ppafR21rJ7DYvGiH73c9ImIUg3WpgaUIOCQNoYa4z5Dei6CVWvWmieWvjke8FZqYMSEf6uO2up/",
30. **"$6$KB$jEdOizEAVqIejbtgX5scm2BA/6juPwGz9yf3FbQ1bHcGRm.sYNlg25ayn/VF8VmHIdApj1G6BrNHpJSm9D8id/",**
32. "$6$KB$1MQg6WZVBTfmwX2SPltKsdEjXziEiMAJ3K8BrxTGvJb3mMmA6lZRhj8pZpF9qemiCkRgzce1lzCq0zBsSNNg8/"
33. };
35. **void substr(char \*dest, char \*src, int start, int length){**
36. memcpy(dest, src + start, length);
37. \*(dest + length) = '**\0**';
38. }
40. **void kernel\_function1(char \*salt\_and\_encrypted){**
41. int x, y, z,a;
42. char salt[7];
44. char plain[7];
45. **char \*enc;**
46. int count = 0;
48. substr(salt, salt\_and\_encrypted, 0, 6);
50. **for(x='A'; x<='M'; x++){**
51. for(y='A'; y<='Z'; y++){
52. for(z=0; z<=99; z++){
53. for(a=0; a<=99;a++){
54. //printf("Instance 1:");
55. **sprintf(plain, "%c%c%02d",x, y, z);**
56. enc = (char \*) crypt(plain, salt);
57. count++;
58. if(strcmp(salt\_and\_encrypted, enc) == 0){
59. printf("#%-8d%s %s**\n**", count, plain, enc);
60. **}**
61. }
62. }
63. }
64. }
65. **printf("%d solutions explored\n", count);**
66. }
67. void kernel\_function2(char \*salt\_and\_encrypted){
68. int x, y, z,a;
69. char salt[7];
71. char plain[7];
72. char \*enc;
73. int count = 0;
75. **substr(salt, salt\_and\_encrypted, 0, 6);**
77. for(x='N'; x<='Z'; x++){
78. for(y='A'; y<='Z'; y++){
79. for(z=0; z<=99; z++){
80. **for(a=0; a<=99;a++){**
81. //printf("Instance 2:");
83. sprintf(plain, "%c%c%02d",x, y, z);
84. enc = (char \*) crypt(plain, salt);
85. **count++;**
86. if(strcmp(salt\_and\_encrypted, enc) == 0){
87. printf("#%-8d%s %s**\n**", count, plain, enc);
88. }
89. }
90. **}**
91. }
92. }
93. printf("%d solutions explored**\n**", count);
94. }

97. int time\_difference(struct timespec \*start, struct timespec \*finish,
98. long long int \*difference) {
99. long long int ds = finish->tv\_sec - start->tv\_sec;
100. **long long int dn = finish->tv\_nsec - start->tv\_nsec;**
102. if(dn < 0 ) {
103. ds--;
104. dn += 1000000000;
105. **}**
106. \*difference = ds \* 1000000000 + dn;
107. return !(\*difference > 0);
108. }
110. **int main(int argc, char\*\* argv) {**
111. struct timespec start, finish;
112. long long int time\_elapsed;
114. clock\_gettime(CLOCK\_MONOTONIC, &start);

117. int size, rank;
118. int i;
120. **MPI\_Init(NULL, NULL);**
121. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
122. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
123. if(size != 3) {
124. if(rank == 0) {
125. **printf("This program needs to run on exactly 3 processes\n");**
126. }
127. } else {
128. if(rank ==0){
130. **int x;**
132. MPI\_Send(&x, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);
133. MPI\_Send(&x, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);


137. } else if(rank==1){
138. int number;
139. MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,
140. **MPI\_STATUS\_IGNORE);**
141. for(i=0;i<n\_passwords;i<i++) {
142. kernel\_function1(encrypted\_passwords[i]);
143. }
144. }
145. **else{**
146. int number;
147. MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,
148. MPI\_STATUS\_IGNORE);
149. for(i=0;i<n\_passwords;i<i++) {
150. **kernel\_function2(encrypted\_passwords[i]);**
151. }
152. }
153. }
154. MPI\_Finalize();
155. **clock\_gettime(CLOCK\_MONOTONIC, &finish);**
156. time\_difference(&start, &finish, &time\_elapsed);
157. printf("Time elapsed was %lldns or %0.9lfs**\n**", time\_elapsed,
158. (time\_elapsed/1.0e9));
160. **return 0;**
161. }

**Output of six digit password with MPI**



Figure 8 Time elapsed for six-digit two initial with four digits

The total time taken by the program is around 29426 seconds (mean of all the time taken by three process) which is 490.43333 minutes and 8.17388883333 hours. It took almost 8 hours to find the required output. I have left the computer in sleep mode without distracting the processor, So, it take small time than expected.

## 3.2 Image Processing

**Code for MPI Image Processing:**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <GL/glut.h>
5. **#include <GL/gl.h>**
6. #include <malloc.h>
7. #include <signal.h>
8. #include <mpi.h>
9. #include <time.h>
10. ***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****
11. *Displays two grey scale images. On the left is an image that has come from an*
12. *image processing pipeline, just after colour thresholding. On the right is*
13. *the result of applying an edge detection convolution operator to the left*
14. *image. This program performs that convolution.*
16. *Things to note:*
17. *- A single unsigned char stores a pixel intensity value. 0 is black, 256 is*
18. *white.*
19. *- The colour mode used is GL\_LUMINANCE. This uses a single number to*
20. ***represent a pixel's intensity. In this case we want 256 shades of grey,***
21. *which is best stored in eight bits, so GL\_UNSIGNED\_BYTE is specified as*
22. *the pixel data type.*

25. ***To compile adapt the code below wo match your filenames:***
26. *mpicc -o Bishalm Bishalm.c -lglut -lGL -lm -lcrypt*
28. *mpirun -n 5 -quiet ./Bishalm*
30. ***Dr Kevan Buckley, University of Wolverhampton, 2018***
31. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
32. #define width 100
33. #define height 72
35. **unsigned char image[], results[width \* height];**
36. int startIndex, endIndex;
38. int time\_difference(struct timespec \*start,
39. struct timespec \*finish,
40. **long long int \*difference) {**
41. long long int ds = finish->tv\_sec - start->tv\_sec;
42. long long int dn = finish->tv\_nsec - start->tv\_nsec;
44. if(dn < 0 ) {
45. **ds--;**
46. dn += 1000000000;
47. }
48. \*difference = ds \* 1000000000 + dn;
49. return !(\*difference > 0);
50. **}**
52. void detect\_edges(unsigned char \*in, unsigned char \*out) {
53. int i;
54. int n\_pixels = (width \* height);
56. for(i=0;i<n\_pixels;i++) {
57. int x, y; // the pixel of interest
58. int b, d, f, h; // the pixels adjacent to x,y used for the calculation
59. int r; // the result of calculate
61. y = i / width;
62. x = i - (width \* y);
64. if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {
65. **results[i] = 0;**
66. } else {
67. b = i + width;
68. d = i - 1;
69. f = i + 1;
70. **h = i - width;**
72. r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)
73. + (in[h] \* -1);
75. **if (r > 0) { // if the result is positive this is an edge pixel**
76. out[i] = 255;
77. } else {
78. out[i] = 0;
79. }
80. **}**
81. }
82. }
84. void tidy\_and\_exit() {
85. **exit(0);**
86. }
88. void sigint\_callback(int signal\_number){
89. printf("**\n**Interrupt from keyboard**\n**");
90. **tidy\_and\_exit();**
91. }
93. static void display() {
94. glClear(GL\_COLOR\_BUFFER\_BIT);
95. **glRasterPos4i(-1, -1, 0, 1);**
96. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);
97. glRasterPos4i(0, -1, 0, 1);
98. glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);
99. glFlush();
100. **}**
102. static void key\_pressed(unsigned char key, int x, int y) {
103. switch(key){
104. case 27: // escape
105. **tidy\_and\_exit();**
106. break;
107. default:
108. printf("**\n**Press escape to exit**\n**");
109. break;
110. **}**
111. }
113. int main(int argc, char \*\*argv) {
114. signal(SIGINT, sigint\_callback);
115. **// printf("image dimensions %dx%d\n", width, height);**
116. // struct timespec start, finish;
117. // long long int difference;
118. int account = 0;
120. **int size, rank;**
121. // clock\_gettime(CLOCK\_MONOTONIC, &start);
122. MPI\_Init(NULL, NULL);
123. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
124. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
125. **if(size != 5) {**
126. if(rank == 0) {
127. printf("This program needs 5 processes**\n**");
128. }
129. } else {
130. **if(rank ==0){**
131. struct timespec start, finish;
132. long long int difference;
133. clock\_gettime(CLOCK\_MONOTONIC, &start);
134. MPI\_Send(&results[0], 1800, MPI\_UNSIGNED\_CHAR, 1, 0, MPI\_COMM\_WORLD);
135. **MPI\_Send(&results[1800], 1800, MPI\_UNSIGNED\_CHAR, 2, 0, MPI\_COMM\_WORLD);**
136. MPI\_Send(&results[3600], 1800, MPI\_UNSIGNED\_CHAR, 3, 0, MPI\_COMM\_WORLD);
137. MPI\_Send(&results[5400], 1800, MPI\_UNSIGNED\_CHAR, 4, 0, MPI\_COMM\_WORLD);
139. MPI\_Recv(&results[0], 1800, MPI\_UNSIGNED\_CHAR, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
140. **MPI\_Recv(&results[1800], 1800,MPI\_UNSIGNED\_CHAR, 2, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**
141. MPI\_Recv(&results[3600], 1800,MPI\_UNSIGNED\_CHAR, 3, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
142. MPI\_Recv(&results[5400], 1800,MPI\_UNSIGNED\_CHAR, 4, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
144. clock\_gettime(CLOCK\_MONOTONIC, &finish);
145. **time\_difference(&start, &finish, &difference);**
146. printf("run lasted %9.5lfs**\n**", difference/1000000000.0);
147. glutInit(&argc, argv);
148. glutInitWindowSize(width \* 2,height);
149. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);
151. glutCreateWindow("6CS005 Image Progessing Courework");
152. glutDisplayFunc(display);
153. glutKeyboardFunc(key\_pressed);
154. glClearColor(0.0, 1.0, 0.0, 1.0);
156. glutMainLoop();
158. tidy\_and\_exit();

161. } else {
162. if(rank == 1){
164. startIndex = 0;
165. **endIndex = 1799;**
166. MPI\_Recv(&results[0], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
167. detect\_edges(image, results);
168. MPI\_Send(&results[0], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);
169. }
170. **else if(rank == 2){**
171. startIndex = 1800;
172. endIndex = 3599;
174. MPI\_Recv(&results[1800], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
175. **detect\_edges(image, results);**
176. MPI\_Send(&results[1800], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);
177. }
178. else if(rank == 3){
179. startIndex = 3600;
180. **endIndex = 5399;**
182. MPI\_Recv(&results[3600], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
183. detect\_edges(image, results);
184. MPI\_Send(&results[3600], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);
186. }
187. else if(rank == 4){
188. startIndex = 5400;
189. endIndex = 7199;
191. MPI\_Recv(&results[5400], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
192. detect\_edges(image, results);
193. MPI\_Send(&results[5400], 1800, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);
194. }
195. **}**
196. }
197. MPI\_Finalize();
198. // clock\_gettime(CLOCK\_MONOTONIC, &finish);
199. // time\_difference(&start, &finish, &difference);
200. **// printf("run lasted %9.5lfs\n", difference/1000000000.0);**
201. return 0;
202. }
204. unsigned char image[] = {255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
205. **255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,0,0,255,255,**
206. **0,255,255,255,0,0,0,0,255,255,0,0,0,0,0,0,255,255,255,**
207. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
208. **255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
209. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,**
210. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,**
211. **255,0,0,255,255,0,0,255,255,0,255,255,0,0,255,0,0,255,255,**
212. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
213. **255,255,255,255,255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,**
214. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
215. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,**
216. **255,255,255,0,0,255,255,0,0,255,255,0,255,0,0,255,255,0,0,**
217. **255,255,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,**
218. **255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,**
219. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
220. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,**
221. **255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,0,0,0,**
222. **0,255,255,255,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,**
223. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,**
224. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
225. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
226. **255,255,0,255,255,255,255,255,0,0,255,255,255,0,0,0,255,255,0,**
227. **0,255,0,0,0,0,255,255,255,0,0,255,255,0,255,255,255,255,255,**
228. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
229. **255,255,255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
230. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
231. **255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,**
232. **0,0,0,255,0,0,255,255,0,0,0,255,255,255,255,0,255,255,255,**
233. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
234. **255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,**
235. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
236. **255,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,**
237. **0,0,0,255,255,255,0,0,255,255,0,0,255,0,0,0,255,255,255,**
238. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
239. **255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,0,**
240. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
241. **255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,0,**
242. **0,255,255,255,255,0,0,255,255,255,255,255,0,0,255,0,0,255,255,**
243. **0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
244. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,**
245. **0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
246. **0,0,0,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,255,**
247. **255,255,255,255,255,0,255,255,255,255,255,0,255,255,255,255,255,0,0,**
248. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
249. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
250. **255,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,**
251. **0,0,0,0,0,0,0,0,0,255,255,0,0,255,255,255,0,0,0,**
252. **255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,0,0,255,**
253. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
254. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
255. **255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,**
256. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,**
257. **255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,255,255,**
258. **255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
259. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
260. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
261. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
262. **255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,**
263. **255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
264. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
265. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
266. **0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
267. **0,0,0,0,255,255,255,255,255,0,255,0,0,255,255,0,0,255,255,**
268. **255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
269. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
270. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
271. **0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,**
272. **0,0,0,0,0,0,0,0,0,255,0,0,255,255,0,0,0,0,255,**
273. **255,255,0,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,**
274. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
275. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
276. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
277. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
278. **255,0,0,0,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,**
279. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
280. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
281. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
282. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
283. **255,255,255,255,255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,**
284. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
285. **255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
286. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
287. **0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,**
288. **0,0,0,0,0,255,255,255,255,255,255,0,0,0,255,255,255,255,255,**
289. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
290. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
291. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
292. **255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,**
293. **0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,255,255,0,255,**
294. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
295. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
296. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
297. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,255,255,255,**
298. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
299. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
300. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,**
301. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
302. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
303. **0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
304. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
305. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
306. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
307. **0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,255,255,255,**
308. **255,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,**
309. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,**
310. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
311. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
312. **0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,255,255,0,**
313. **0,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,0,0,0,**
314. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
315. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
316. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
317. **0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,0,**
318. **255,255,255,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,255,**
319. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
320. **255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
321. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
322. **0,0,0,0,0,0,0,255,255,0,0,255,255,255,255,0,0,255,255,**
323. **255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,255,255,0,0,**
324. **0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
325. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,**
326. **255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
327. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,**
328. **255,0,0,255,255,255,255,255,255,0,255,255,255,255,0,0,255,255,255,**
329. **255,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,**
330. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
331. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
332. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,**
333. **255,0,0,255,255,255,0,0,255,255,255,255,255,255,0,255,255,255,255,**
334. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,**
335. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
336. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
337. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
338. **255,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,**
339. **255,255,255,255,255,0,0,0,0,0,255,255,255,0,0,0,0,0,0,**
340. **0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
341. **255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,**
342. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
343. **0,0,255,255,255,255,0,0,255,255,255,255,0,255,255,255,255,255,255,**
344. **255,255,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,**
345. **0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,**
346. **0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
347. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,**
348. **255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,0,**
349. **255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,**
350. **255,255,255,255,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,**
351. **0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,**
352. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
353. **255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,0,0,0,255,**
354. **255,255,255,0,0,255,255,255,255,255,255,255,0,255,255,255,0,0,255,**
355. **255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,255,**
356. **255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,**
357. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
358. **255,255,0,0,0,255,255,255,255,0,255,255,255,255,255,0,255,255,255,**
359. **255,0,0,0,0,255,255,255,255,0,255,255,255,255,255,255,0,0,255,**
360. **255,255,255,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
361. **0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
362. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
363. **0,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,**
364. **255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,**
365. **255,255,0,0,255,255,255,255,0,0,0,255,255,255,255,255,0,0,0,**
366. **0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,**
367. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
368. **0,0,0,0,255,255,255,255,0,255,255,255,0,0,0,255,255,255,0,**
369. **0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,**
370. **0,0,255,255,255,255,255,0,0,0,255,255,255,0,0,0,0,0,0,**
371. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,**
372. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
373. **0,0,0,0,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,**
374. **0,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,**
375. **0,0,255,255,255,0,0,255,255,255,255,255,0,0,0,255,255,255,0,**
376. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
377. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
378. **0,0,0,0,0,0,0,255,255,255,0,0,0,0,255,255,255,255,0,**
379. **255,255,255,255,0,255,255,255,255,0,0,0,0,255,255,255,255,255,255,**
380. **255,255,0,0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,**
381. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
382. **0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,**
383. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
384. **255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,0,0,0,255,**
385. **255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,0,0,255,**
386. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
387. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,**
388. **0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,255,255,**
389. **255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,**
390. **255,255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,**
391. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
392. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,**
393. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,**
394. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,**
395. **255,255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,255,255,**
396. **255,255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
397. **0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,**
398. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
399. **255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
400. **0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,255,255,255,255,**
401. **0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
402. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,**
403. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
404. **0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,255,255,255,255,**
405. **0,255,255,255,255,0,0,0,255,255,255,255,0,0,255,255,255,255,0,**
406. **0,255,255,255,255,0,0,0,255,255,0,0,0,0,0,0,0,0,0,**
407. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
408. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
409. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,255,255,0,**
410. **0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,255,0,0,**
411. **255,255,255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,**
412. **0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,**
413. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
414. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
415. **255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,0,0,0,255,**
416. **255,255,255,255,255,255,255,255,255,0,0,0,255,0,0,0,0,0,0,**
417. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
418. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
419. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
420. **0,0,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,**
421. **255,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
422. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
423. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
424. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
425. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,255,**
426. **255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,255,0,0,0,**
427. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
428. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
429. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
430. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,255,**
431. **255,255,255,0,0,255,255,255,255,255,255,0,0,0,0,255,255,255,0,**
432. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
433. **255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
434. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
435. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
436. **255,255,0,0,0,255,255,255,0,0,0,255,255,255,255,255,0,0,0,**
437. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
438. **0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,**
439. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
440. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
441. **0,0,0,0,255,255,255,255,0,255,255,255,255,0,0,0,255,255,255,**
442. **255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
443. **0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
444. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
445. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
446. **0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,0,**
447. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
448. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,**
449. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
450. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
451. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
452. **255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
453. **0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,**
454. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
455. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
456. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
457. **0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,**
458. **0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,**
459. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
460. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
461. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
462. **0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,**
463. **0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,**
464. **255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
465. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
466. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
467. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
468. **0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,**
469. **255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
470. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
471. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
472. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
473. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,**
474. **255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,**
475. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
476. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
477. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
478. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,**
479. **255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,**
480. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
481. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
482. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
483. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,**
484. **255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,**
485. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
486. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
487. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
488. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,**
489. **255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,**
490. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
491. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
492. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
493. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
494. **0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,**
495. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
496. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
497. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
498. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
499. **0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,**
500. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
501. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
502. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
503. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
504. **0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,**
505. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
506. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
507. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
508. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
509. **0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,**
510. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
511. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
512. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
513. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
514. **0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,**
515. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
516. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
517. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
518. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
519. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,**
520. **255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
521. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
522. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
523. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
524. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
525. **0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
526. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
527. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
528. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
529. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
530. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
531. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
532. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
533. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
534. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
535. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
536. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
537. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
538. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
539. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
540. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
541. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
542. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
543. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
544. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
545. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
546. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
547. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
548. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
549. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
550. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
551. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
552. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
553. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
554. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
555. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
556. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
557. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
558. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
559. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
560. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
561. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
562. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
563. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
564. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
565. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
566. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
567. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
568. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
569. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
570. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
571. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
572. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
573. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
574. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
575. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
576. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
577. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
578. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
579. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
580. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
581. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**
582. **0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0**
583. **};**

**Explanation of MPI Image Processing:**

The main aim of this code is to know how computer stores and manipulate two-dimensional array of cells called image or picture. Picture is made up of pixels which is made up of horizontal and vertical coordinate x and y respectively.

Two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameter CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. In MPI, we must declare to variable size and time and MPI library is initialized by MPI\_Init datatype. MPI\_Comm\_size() function is used to identify size of communicator and MPI\_Comm\_rank() function is used to identify the rank of the calling process. The program only executes if there are exactly three process. The MPI program MPI\_Send() is used to do blocking send to other rank from the current rank. MPI\_Send has six parameters address of send buffer, count in send buffer, datatype of each send buffer element, rank of destination, message tag and communicator We have used MPI\_Send() to send the work to the rank 1, 2, 3 and 4 from rank 0. We used MPI\_Unsigned\_char datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator. We have also sent 1800 count on each rank to four rank which makes the total of 7200 which is exactly the size of picture.

Similarly, MPI\_Recv() is used to receive for sent message from another rank to the current rank and it has 7 paraemters address of send buffer,Count in send buffer,datatype of each send buffer element,Rank Source of the sender, message tag,communicaton, and Status

We have used MPI\_Recv() to receive the work from rank 0. We used MPI\_Unsigned\_char datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator. We ignored the status of the receive with MPI\_STATUS\_IGNORE keyword and count as 1800. We have used startIndex and endIndex in each of the rank which is shown in table below:

|  |  |  |  |
| --- | --- | --- | --- |
| S.N. | Rank | StartIndex | endIndex |
| 1 | Rank1 | 0 | 1799 |
| 2 | Rank2 | 1800 | 3599 |
| 3 | Rank3 | 3600 | 5399 |
| 4 | Rank4 | 5400 | 7199 |

Table 10 StartIndex and End index for all rank

From all rank1, 2, 3,and 4 we have called the detect\_edges() function after receiving the message from rank 0.

detect\_edges function requires the three-parameter image, result and argument. Image is denoted by in pointer, result is denoted by out pointer and argument is denoted by arg pointer in detect-edges function. First of all, the number of pixels is calculated by multiplying width and height of picture which is set to 100 and 72 respectively at the top of the program. Our program uses convolution formula to detect the edge of the given picture. To detect the edge x, y pixels are taken from the image and defined as the

y = i / width;

x = i - (width \* y);

The value of result can be calculated by using the value obtaining formula given by convolution theorem:

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

If the value of r is greater than 0 that is positive, edge of output result is set to white color (255) and if value is not equal to zero then the output result will be set to black color. Since we have worked on different processes which needs to be combined together to get the desired output image. To send the work process to rank 0 we used MPI\_Send to send the finished work to rank 0 from rank1, rank2, rank3 and rank4. MPI\_Receive() function is used in rank0 to receive all the work After completion, the total time taken by the program is calculated by the function named time\_difference() function.The image is then showed to the windows with the help of utility kit GLUT. Different function of GLUT like glutInit, glutInitWindowSize, glutInitDisplayMode, glutCreateWindow, glutDisplayFunc, glutKeyboardFunc, glClearColor, glutMainLoop etc. are used to display the image. Since we have used MPI library specification, we must use MPI\_Finalize() function to terminates the execution environment of MPI.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (MPI) |
| 1 | 0.000065933s | 0.00144s |
| 2 | 0.000115084s | 0.0006s |
| 3 | 0.000064197s | 0.00035s |
| 4 | 0.000065573s | 0.00034s |
| 5 | 0.000066473s | 0.00030s |
| 6 | 0.000122975s | 0.00316s |
| 7 | 0.000094871s | 0.00036s |
| 8 | 0.000068225s | 0.00045s |
| 9 | 0.000063991s | 0.00075s |
| 10 | 0.000063396s | 0.00051s |
|  | Mean Running time for original program: 0.0000790718s | Mean Running time for MPI program: 0.000826s |

Table 11 Comparison between original and MPI version of Image processing

**Write a short analysis of the results:**

After seeing the result, the mean running time MPI program took0.000826s whereas the mean Running time for original program is only 0.0000790718s. The MPI program for image processing took more time than the original program which seems to be abnormal in our case. The time taken by the original program is one thousand fractions faster than the program which uses MPI. This may be due to the use of small image with only 7200 pixels. The program tries to solve the small image parallelly to detect the edges overhead has occurred. So, it takes more time than excepted. Moreover, the MPI program needs some time to send the work from master to other processes.

## 3.3 Linear Regression

**Code for MPI Linear Regression:**

1. #include <stdio.h>
2. #include <math.h>
3. #include <mpi.h>
4. #include <time.h>
6. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
7. *\* This program takes an initial estimate of m and c and finds the associated*
8. *\* rms error. It is then as a base to generate and evaluate 8 new estimates,*
9. *\* which are steps in different directions in m-c space. The best estimate is*
10. ***\* then used as the base for another iteration of "generate and evaluate". This***
11. *\* continues until none of the new estimates are better than the base. This is*
12. *\* a gradient search for a minimum in mc-space.*
13. *\**
14. *\* To compile:*
15. ***\* mpicc Mpilinearbishal.c -o Mpilinearbishal -lm***
16. *\**
17. *\* To run:*
18. *\* mpirun -n 9 ./Mpilinearbishal*
19. *\*To run 10 times"*
20. ***\* mpirun -n 9 ./mr.py ./Mpilinearbishal***
21. *\**
22. *\* Dr Kevan Buckley, University of Wolverhampton, 2018*
23. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
25. **typedef struct point\_t**
26. {
27. double x;
28. double y;
29. } point\_t;
31. int n\_data = 1000;
32. point\_t data[];
34. double residual\_error (double x, double y, double m, double c)
35. **{**
36. double e = (m \* x) + c - y;
37. return e \* e;
38. }
40. **double rms\_error (double m, double c)**
41. {
42. int i;
43. double mean;
44. double error\_sum = 0;
46. for (i = 0; i < n\_data; i++)
47. {
48. error\_sum += residual\_error (data[i].x, data[i].y, m, c);
49. }
51. mean = error\_sum / n\_data;
53. return sqrt (mean);
54. }
55. **int time\_difference(struct timespec \*start, struct timespec \*finish,**
56. long long int \*difference) {
57. long long int ds = finish->tv\_sec - start->tv\_sec;
58. long long int dn = finish->tv\_nsec - start->tv\_nsec;
60. **if(dn < 0 ) {**
61. ds--;
62. dn += 1000000000;
63. }
64. \*difference = ds \* 1000000000 + dn;
65. **return !(\*difference > 0);**
66. }
67. int main () {

70. **struct timespec start, finish;**
71. long long int time\_elapsed;
72. clock\_gettime(CLOCK\_MONOTONIC, &start);

75. **int rank, size;**
76. int i;
77. double bm = 1.3;
78. double bc = 10;
79. double be;
80. **double dm[8];**
81. double dc[8];
82. double e[8];
83. double step = 0.01;
84. double best\_error = 999999999;
85. **int best\_error\_i;**
86. int minimum\_found = 0;
87. double pError = 0;
88. double baseMC[2];
90. **double om[] = { 0, 1, 1, 1, 0, -1, -1, -1 };**
91. double oc[] = { 1, 1, 0, -1, -1, -1, 0, 1 };

94. MPI\_Init (NULL, NULL);
95. **MPI\_Comm\_size (MPI\_COMM\_WORLD, &size);**
96. MPI\_Comm\_rank (MPI\_COMM\_WORLD, &rank);
98. be = rms\_error (bm, bc);
100. **if (size != 9)**
101. {
102. if (rank == 0)
103. {
104. printf
105. **("This program is made for run with only 9 processes.\n");**
106. return 0;
107. }
108. }
110. **while (!minimum\_found)**
111. {
113. if (rank != 0)
114. {
115. **i = rank - 1;**
116. dm[i] = bm + (om[i] \* step);
117. dc[i] = bc + (oc[i] \* step);
118. pError = rms\_error (dm[i], dc[i]);
120. **MPI\_Send (&pError, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);**
121. MPI\_Send (&dm[i], 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);
122. MPI\_Send (&dc[i], 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

125. **MPI\_Recv (&bm, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**
126. MPI\_Recv (&bc, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
127. MPI\_Recv (&minimum\_found, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
128. }
129. else
130. **{**
131. for (i = 1; i < size; i++)
132. {
133. MPI\_Recv (&pError, 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
134. MPI\_Recv (&dm[i-1], 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);
135. **MPI\_Recv (&dc[i-1], 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**
136. if (pError < best\_error)
137. {
138. best\_error = pError;
139. best\_error\_i = i - 1;
141. }
142. }
144. if (best\_error < be)
145. **{**
146. be = best\_error;
147. bm = dm[best\_error\_i];
148. bc = dc[best\_error\_i];
149. }
150. **else**
151. {
152. minimum\_found = 1;
153. }
155. **for (i = 1; i < size; i++)**
156. {
157. MPI\_Send (&bm, 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);
158. MPI\_Send (&bc, 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);
159. MPI\_Send (&minimum\_found, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);
160. **}**
161. }
162. }

165. **if(rank==0) {**
166. printf ("minimum m,c is %lf,%lf with error %lf**\n**", bm, bc, be);
167. clock\_gettime(CLOCK\_MONOTONIC, &finish);
168. time\_difference(&start, &finish, &time\_elapsed);
169. printf("Time elapsed : %lldns or %0.9lfs**\n**", time\_elapsed,
170. **(time\_elapsed/1.0e9));**
171. }
173. MPI\_Finalize();
174. return 0;
175. **}**
177. point\_t data[] = {
178. {72.12,100.78},{65.40,107.86},{82.27,131.60},{82.31,122.34},
179. {89.41,121.50},{71.37,113.51},{82.62,112.38},{69.57,102.96},
180. **{65.38,99.27},{84.50,138.85},{87.18,114.17},{73.03,109.21},**
181. {67.26,102.06},{72.25,113.23},{61.28,101.59},{41.60,84.24},
182. {40.14,57.03},{15.24,45.58},{61.88,89.90},{34.89,72.77},
183. { 8.91,36.34},{30.45,46.18},{67.93,89.35},{68.82,112.80},
184. {63.96,99.32},{32.36,56.12},{42.20,63.66},{24.47,60.75},
185. **{ 1.96,28.62},{41.42,68.41},{34.49,73.14},{ 8.03,22.13},**
186. {80.55,117.79},{85.54,130.80},{68.99,103.13},{99.32,144.79},
187. {91.71,153.61},{71.17,108.40},{85.28,120.11},{99.52,128.68},
188. {13.24,31.67},{ 5.19,40.15},{ 9.84,57.36},{29.42,54.01},
189. {89.68,126.25},{29.45,41.30},{79.63,132.59},{71.88,107.31},
190. **{20.05,48.38},{40.98,54.11},{56.55,63.61},{77.22,114.17},**
191. {63.86,88.10},{92.93,134.84},{56.84,101.20},{34.31,71.18},
192. {93.89,116.43},{38.02,63.78},{61.25,94.71},{71.02,103.42},
193. {95.05,142.82},{96.24,133.50},{19.50,50.92},{41.14,70.59},
194. {91.49,134.05},{54.05,98.31},{36.59,68.48},{91.14,130.45},
195. **{44.76,88.98},{77.28,138.16},{64.80,96.33},{43.25,70.08},**
196. {55.55,95.70},{ 3.77,39.03},{ 3.23,44.69},{86.72,127.42},
197. {84.62,131.54},{26.13,71.24},{61.22,98.22},{53.90,96.07},
198. {64.81,109.35},{91.66,116.79},{53.65,104.81},{38.42,66.16},
199. {62.33,112.41},{ 7.41,29.86},{41.59,57.59},{56.49,91.60},
200. **{15.94,42.82},{97.46,140.29},{57.17,85.11},{26.94,45.86},**
201. {73.14,96.37},{18.61,60.58},{15.69,44.16},{20.79,33.86},
202. {65.02,106.03},{38.09,72.71},{87.15,116.68},{77.45,123.08},
203. {90.47,126.33},{26.80,44.96},{75.94,119.76},{33.83,69.11},
204. {63.59,103.98},{38.05,72.36},{68.28,110.76},{ 3.34,54.22},
205. **{45.40,92.84},{78.37,113.49},{27.11,46.46},{32.32,68.44},**
206. {20.97,30.90},{37.92,75.11},{96.85,130.96},{69.40,95.17},
207. { 3.29,30.06},{64.41,103.44},{15.80,52.64},{61.76,97.79},
208. { 1.62,33.98},{29.03,58.02},{18.74,34.93},{25.41,73.73},
209. {28.78,65.94},{14.64,50.31},{82.85,133.70},{41.62,90.32},
210. **{99.28,144.95},{90.16,133.18},{40.45,77.72},{ 1.79,50.44},**
211. {31.80,62.71},{26.30,40.89},{47.57,83.15},{17.78,44.90},
212. {69.48,93.13},{87.98,126.95},{69.84,106.00},{37.06,61.61},
213. {90.65,133.97},{10.73,46.60},{38.84,79.90},{ 4.75,33.89},
214. {48.99,89.31},{ 2.51,47.09},{34.99,86.40},{29.79,54.52},
215. **{91.30,133.72},{74.12,122.86},{90.93,141.88},{51.14,89.93},**
216. {84.53,142.49},{26.84,58.79},{ 6.95,20.98},{49.80,85.14},
217. {22.82,57.02},{44.08,89.32},{22.28,48.72},{21.12,50.68},
218. {65.69,93.93},{27.84,39.97},{ 1.92,40.39},{ 9.36,33.54},
219. {88.10,123.02},{18.15,63.84},{21.80,39.76},{64.42,101.03},
220. **{ 2.23,22.52},{55.68,99.56},{37.55,87.77},{74.23,104.87},**
221. {11.96,37.30},{23.60,45.84},{11.13,34.32},{ 9.05,48.79},
222. {56.11,100.21},{19.31,54.44},{ 6.27,16.17},{64.65,101.39},
223. {50.25,77.59},{69.33,95.12},{47.52,87.79},{28.97,65.98},
224. {71.56,95.30},{19.71,41.47},{57.66,96.65},{41.07,74.10},
225. **{35.08,79.46},{40.80,87.01},{ 0.31,19.82},{90.78,111.55},**
226. {34.39,72.03},{99.97,139.40},{30.86,73.03},{14.37,50.15},
227. { 6.11,42.76},{21.75,80.30},{89.94,127.56},{10.86,42.40},
228. {13.07,42.98},{84.47,147.14},{83.44,132.18},{32.24,63.57},
229. {66.93,102.41},{34.48,68.96},{ 3.46,22.82},{94.84,130.83},
230. **{49.41,107.26},{71.64,99.82},{47.28,80.62},{39.17,68.77},**
231. {58.05,108.35},{69.27,109.81},{47.64,73.34},{34.64,73.15},
232. {22.86,46.34},{37.76,66.19},{ 3.12,39.11},{60.59,111.05},
233. {91.99,122.76},{96.60,138.86},{ 3.58,23.35},{22.81,60.18},
234. {13.93,21.32},{69.51,106.41},{19.57,43.39},{79.11,115.68},
235. **{80.89,124.36},{44.42,57.78},{33.28,73.04},{21.45,49.88},**
236. {70.57,113.77},{45.63,65.60},{55.99,72.21},{21.62,41.47},
237. {61.74,98.99},{ 9.30,29.77},{75.32,106.74},{27.97,73.44},
238. {74.77,115.98},{42.93,82.67},{92.32,138.05},{25.55,64.34},
239. { 0.48,23.51},{79.52,111.52},{52.83,70.58},{51.45,87.28},
240. **{62.72,90.41},{ 4.16,40.60},{70.13,115.25},{55.96,97.34},**
241. {93.88,154.09},{46.21,90.04},{34.75,51.46},{54.45,89.56},
242. {80.69,129.36},{45.14,73.00},{47.34,85.69},{70.16,118.02},
243. { 4.26,17.14},{61.56,98.04},{15.95,28.56},{74.06,118.48},
244. {65.29,99.71},{19.08,55.64},{37.82,72.36},{58.22,103.93},
245. **{50.52,82.15},{26.25,60.91},{97.77,123.91},{39.13,68.03},**
246. {15.09,41.88},{32.61,61.64},{11.23,22.85},{61.92,98.02},
247. {73.63,126.32},{35.12,54.74},{12.98,42.69},{83.87,128.60},
248. {45.65,78.81},{42.85,90.57},{76.74,117.53},{19.05,49.60},
249. {69.03,104.16},{23.66,54.97},{52.85,85.94},{82.07,128.27},
250. **{74.77,111.22},{95.04,136.69},{40.49,49.53},{ 4.16,28.40},**
251. { 7.69,51.29},{29.37,80.82},{86.06,122.19},{ 3.92,23.24},
252. {62.76,108.89},{27.12,54.24},{10.24,33.84},{79.86,107.97},
253. {57.09,85.27},{10.29,54.38},{53.50,82.98},{12.83,50.29},
254. { 2.09,13.69},{88.73,135.16},{42.72,87.10},{40.20,91.88},
255. **{40.10,76.49},{80.22,133.65},{57.55,93.99},{29.34,69.08},**
256. { 2.90,41.26},{44.60,82.03},{47.93,89.05},{98.17,123.11},
257. {17.21,45.91},{42.37,79.83},{90.89,119.42},{ 7.81,36.64},
258. {76.14,123.86},{47.79,83.40},{95.27,144.30},{44.13,98.20},
259. {19.97,37.36},{90.66,131.96},{75.41,117.80},{57.14,107.91},
260. **{25.92,41.69},{90.86,130.36},{44.78,79.02},{23.00,29.10},**
261. {91.67,118.13},{26.55,51.18},{41.60,74.91},{ 0.39, 6.79},
262. {86.31,102.08},{20.43,37.80},{ 5.39,28.65},{12.63,24.33},
263. {22.60,42.79},{ 1.77,14.54},{74.10,113.64},{54.46,87.67},
264. {18.64,49.32},{93.97,116.30},{42.62,87.04},{13.37,30.16},
265. **{74.50,104.62},{18.28,67.85},{76.98,107.84},{25.89,57.35},**
266. {13.52,42.87},{61.26,97.78},{ 5.97,31.34},{91.99,137.43},
267. {20.38,58.23},{ 9.59,31.56},{79.41,126.40},{89.90,134.36},
268. {73.18,111.44},{61.51,111.41},{99.96,147.82},{72.55,113.52},
269. {66.21,110.93},{36.47,59.41},{65.58,93.39},{24.93,51.71},
270. **{58.00,95.89},{49.83,83.52},{53.35,89.98},{83.97,129.85},**
271. {57.33,106.86},{53.94,98.13},{98.02,144.26},{47.28,72.52},
272. {45.48,100.70},{80.69,147.66},{96.14,140.01},{82.69,120.80},
273. {79.73,136.89},{11.42,27.51},{88.91,138.59},{25.53,51.26},
274. { 2.49,37.14},{63.89,93.28},{90.96,138.02},{15.27,53.03},
275. **{25.39,51.31},{31.77,55.54},{88.25,124.46},{67.66,108.26},**
276. {90.23,112.02},{17.40,43.85},{78.38,137.07},{96.28,149.45},
277. {77.38,120.54},{56.49,107.27},{99.00,141.67},{36.35,58.18},
278. {97.41,132.64},{15.03,48.28},{42.48,81.20},{62.95,105.32},
279. {99.76,147.11},{85.18,140.95},{99.23,131.84},{21.09,44.44},
280. **{45.12,75.22},{80.36,119.71},{61.37,84.74},{82.64,128.58},**
281. {70.34,108.16},{83.63,116.26},{47.73,67.57},{17.56,48.42},
282. {23.26,42.12},{41.81,82.17},{18.48,33.63},{39.11,70.14},
283. {84.20,123.97},{67.20,113.97},{52.74,87.79},{81.66,131.54},
284. {45.90,93.69},{20.82,34.77},{86.35,122.38},{78.93,106.82},
285. **{10.56,44.66},{51.20,104.61},{93.79,131.97},{15.71,43.06},**
286. {99.16,156.47},{90.70,135.27},{41.85,77.91},{73.41,106.66},
287. {57.51,108.55},{53.06,115.27},{25.72,67.45},{ 8.03,27.74},
288. {57.91,101.56},{35.87,57.47},{98.33,145.81},{50.96,76.84},
289. {57.86,102.10},{17.21,44.21},{95.62,154.59},{76.92,114.77},
290. **{25.32,60.66},{43.60,68.34},{42.68,73.98},{60.36,84.81},**
291. { 9.06,42.91},{ 4.16,18.44},{54.14,97.87},{ 4.87,35.92},
292. {75.38,112.62},{41.37,68.92},{88.16,163.96},{16.79,41.87},
293. { 9.77,40.62},{69.66,125.12},{70.35,118.66},{71.99,97.87},
294. {63.66,111.29},{ 2.01,19.46},{64.63,122.89},{48.39,84.19},
295. **{28.15,64.69},{46.17,83.91},{25.12,45.94},{82.23,118.70},**
296. {57.69,95.98},{24.42,62.91},{15.81,35.58},{75.28,106.87},
297. {95.74,133.25},{67.78,107.42},{80.89,128.72},{10.39,38.37},
298. {15.31,35.73},{61.45,110.46},{11.15,44.99},{30.80,63.26},
299. {84.29,122.39},{29.17,47.34},{80.68,138.44},{81.17,117.86},
300. **{ 8.47,32.78},{41.26,74.09},{43.50,71.18},{34.48,68.61},**
301. {30.63,68.05},{88.63,137.28},{71.56,116.97},{21.03,39.12},
302. {88.20,116.24},{ 8.52,30.24},{95.79,137.27},{78.66,104.62},
303. {72.44,94.21},{71.60,106.34},{72.11,114.18},{34.50,59.18},
304. {22.85,60.95},{18.43,40.91},{69.24,119.69},{91.84,142.06},
305. **{34.41,69.95},{95.06,136.92},{67.93,100.93},{46.96,71.82},**
306. {63.92,102.14},{ 1.62,29.66},{95.24,133.60},{43.10,80.88},
307. {21.83,73.25},{35.01,62.42},{20.05,55.19},{18.64,45.92},
308. {40.28,75.26},{34.54,63.38},{84.74,117.68},{90.38,144.87},
309. { 9.91,24.87},{62.97,102.14},{34.40,79.20},{67.34,89.48},
310. **{48.53,85.13},{24.57,51.59},{81.95,117.78},{22.23,49.77},**
311. {75.86,125.20},{60.45,99.78},{19.93,35.57},{48.62,78.46},
312. {88.49,120.71},{13.33,40.67},{52.03,93.38},{38.43,80.28},
313. { 2.56,17.00},{18.39,58.10},{58.81,88.08},{75.76,96.69},
314. {69.78,98.83},{96.47,146.81},{47.32,79.89},{21.90,46.54},
315. **{52.39,83.38},{75.49,107.96},{50.14,80.51},{41.54,73.80},**
316. {76.07,117.48},{27.00,73.59},{81.59,122.88},{21.74,39.55},
317. {60.05,105.04},{75.68,102.72},{40.41,79.01},{ 0.32,24.82},
318. {50.06,106.14},{98.69,139.50},{64.17,109.26},{42.74,78.53},
319. {39.52,71.78},{55.14,97.37},{25.19,39.08},{99.31,142.63},
320. **{67.50,91.86},{90.92,152.17},{81.99,129.38},{77.28,124.08},**
321. {29.38,69.15},{ 3.81,41.93},{ 9.72,41.83},{25.75,53.09},
322. {57.28,85.11},{69.50,116.90},{20.00,51.46},{63.00,72.32},
323. {67.06,102.20},{37.85,64.86},{81.40,114.28},{13.32,58.41},
324. {67.21,103.77},{63.73,109.66},{91.43,141.66},{54.83,88.07},
325. **{68.03,112.67},{ 0.51,27.76},{ 2.17,38.05},{36.26,66.58},**
326. {72.67,116.52},{98.28,136.37},{85.27,128.64},{90.26,136.47},
327. {60.31,95.24},{32.77,58.94},{ 3.52,24.75},{15.98,45.49},
328. {94.25,145.90},{ 8.13,29.89},{61.13,81.38},{44.14,77.64},
329. {63.53,100.35},{49.35,97.92},{ 4.98,32.12},{25.53,57.45},
330. **{ 8.63,41.62},{24.23,56.27},{93.30,137.92},{43.72,71.72},**
331. {54.15,89.12},{ 3.42,36.34},{57.75,85.68},{51.90,87.74},
332. {85.14,137.82},{99.27,173.87},{82.53,124.94},{15.38,44.42},
333. {66.66,108.56},{64.12,99.41},{39.08,73.77},{25.42,58.25},
334. { 1.29,36.39},{98.72,148.84},{70.09,112.06},{ 8.51,27.00},
335. **{85.92,124.74},{88.32,127.04},{51.79,74.58},{36.46,62.45},**
336. {49.29,85.33},{14.06,30.58},{24.83,34.82},{42.85,87.06},
337. {34.47,76.96},{59.16,90.44},{ 1.02,32.32},{61.80,108.22},
338. {72.52,95.83},{65.40,99.49},{53.32,93.79},{74.22,117.61},
339. {53.86,88.31},{39.84,80.11},{79.28,117.86},{34.57,76.73},
340. **{21.69,55.55},{99.87,129.34},{72.12,108.86},{75.08,106.64},**
341. {70.71,106.00},{18.35,67.45},{37.42,66.71},{ 0.70, 9.02},
342. {56.79,86.75},{74.04,100.45},{53.40,82.23},{42.13,70.45},
343. {82.43,123.55},{91.65,131.55},{94.99,153.70},{62.14,84.17},
344. {99.71,151.07},{33.24,73.77},{48.87,76.91},{68.57,118.95},
345. **{14.28,46.22},{18.17,41.01},{95.93,133.32},{ 5.06,33.23},**
346. {57.58,95.47},{18.71,39.10},{90.19,136.73},{26.98,50.08},
347. {11.36,26.14},{62.70,98.59},{49.32,80.54},{99.97,149.27},
348. {83.40,132.00},{25.30,48.62},{79.25,117.83},{81.09,109.23},
349. {31.46,51.02},{14.26,32.26},{33.53,52.63},{ 9.42,47.16},
350. **{67.40,109.90},{18.56,32.79},{34.51,75.14},{49.00,77.38},**
351. {15.69,50.80},{23.09,40.32},{32.03,67.86},{13.60,40.35},
352. {19.21,60.16},{78.56,111.57},{80.72,131.02},{50.19,79.64},
353. {55.60,81.78},{ 6.37,43.37},{42.78,74.85},{60.48,113.67},
354. {44.44,89.27},{54.02,90.24},{73.51,101.74},{16.41,56.73},
355. **{70.94,104.90},{32.03,66.91},{13.12,49.71},{50.16,85.64},**
356. {41.31,68.88},{69.25,123.25},{24.97,69.28},{40.80,86.30},
357. {32.28,67.01},{90.77,142.80},{66.77,104.70},{24.06,56.12},
358. {49.16,89.52},{46.10,95.56},{51.79,94.01},{56.11,100.66},
359. {88.49,126.71},{ 1.28,21.35},{35.55,64.10},{18.79,29.74},
360. **{ 5.40,40.02},{92.32,129.89},{21.13,47.05},{ 5.14,32.16},**
361. {60.89,104.41},{43.45,76.07},{98.91,160.53},{99.31,155.80},
362. {74.71,121.53},{62.33,98.98},{58.66,101.10},{51.51,93.03},
363. {51.69,90.42},{19.47,31.22},{85.75,108.87},{64.20,100.48},
364. {96.60,142.66},{67.99,102.48},{68.37,120.07},{29.81,44.77},
365. **{96.55,142.74},{30.59,43.25},{73.94,108.44},{49.77,88.88},**
366. {59.48,98.21},{41.21,61.86},{38.63,83.41},{86.98,140.40},
367. {93.34,134.69},{87.92,119.52},{40.93,61.87},{ 2.43,30.68},
368. {50.74,71.81},{37.13,52.43},{ 1.50,22.18},{99.06,143.48},
369. { 1.67,27.67},{ 0.18,10.50},{54.13,77.05},{46.19,88.91},
370. **{91.13,144.49},{ 8.95,28.33},{85.69,122.61},{50.30,95.60},**
371. {48.63,103.49},{67.99,100.19},{69.21,112.13},{11.26,34.99},
372. {25.78,58.73},{84.35,112.36},{46.80,79.68},{69.54,117.99},
373. {40.30,74.33},{79.97,118.95},{23.28,55.71},{32.62,78.92},
374. {21.86,37.01},{ 5.07,22.57},{94.41,146.15},{40.14,60.81},
375. **{95.80,125.35},{91.34,131.68},{72.55,113.56},{40.13,71.59},**
376. {98.06,145.27},{90.55,144.08},{71.26,121.81},{33.85,71.13},
377. {85.74,142.63},{57.93,91.78},{ 7.63,39.30},{83.72,128.26},
378. {10.89,46.78},{39.79,66.98},{98.84,146.32},{84.62,123.91},
379. {23.16,31.94},{86.36,134.79},{44.19,63.74},{ 0.39,24.19},
380. **{64.22,96.97},{66.47,103.78},{ 1.73,17.52},{22.25,36.77},**
381. {31.88,59.39},{15.60,30.03},{16.08,41.91},{83.11,129.19},
382. {72.61,122.52},{19.02,41.06},{56.90,87.53},{65.85,97.02},
383. {81.40,120.35},{64.90,104.44},{73.35,119.00},{ 8.49,40.31},
384. {31.20,65.32},{28.29,75.05},{72.51,120.90},{20.42,48.84},
385. **{71.46,111.59},{33.98,50.46},{72.48,111.29},{75.56,113.00},**
386. {58.65,95.16},{23.66,44.95},{95.08,139.46},{80.12,115.20},
387. {67.77,101.97},{56.06,99.08},{99.03,138.47},{48.26,74.79},
388. {25.95,39.30},{85.20,137.70},{69.31,104.19},{86.19,122.91},
389. {37.99,87.47},{72.06,116.90},{ 5.66,28.92},{27.77,52.05},
390. **{31.89,60.32},{18.01,48.92},{37.21,65.49},{73.76,107.20},**
391. { 0.32,-0.71},{93.75,133.48},{69.11,109.63},{11.01,55.84},
392. {43.48,73.99},{20.76,57.44},{75.50,105.00},{98.74,150.46},
393. {40.75,90.93},{61.67,103.30},{93.48,155.96},{35.52,61.62},
394. {32.30,78.52},{28.92,49.61},{60.97,87.11},{13.59,47.58},
395. **{ 9.43,26.07},{58.00,107.90},{99.86,151.90},{34.01,57.82},**
396. {39.02,59.14},{33.64,74.99},{ 2.28,20.21},{55.00,90.93},
397. {55.77,85.94},{79.17,134.03},{63.16,106.70},{17.58,32.28},
398. {24.29,34.68},{83.91,132.35},{96.44,129.86},{61.95,93.66},
399. {14.86,25.10},{15.53,33.29},{15.69,42.47},{80.60,126.11},
400. **{16.01,46.33},{26.54,74.55},{ 2.67,37.10},{74.63,96.98},**
401. {38.06,59.99},{56.59,96.87},{78.88,120.95},{87.56,121.75},
402. {73.54,119.27},{16.84,44.09},{44.24,89.36},{76.02,123.64},
403. {98.41,115.45},{12.11,48.19},{30.70,60.41},{55.51,100.49},
404. { 0.26,37.11},{83.43,124.44},{49.92,111.30},{65.55,99.48},
405. **{77.61,119.44},{62.44,95.52},{21.80,61.06},{20.99,60.54},**
406. {93.10,129.45},{54.96,91.05},{10.22,48.48},{66.77,108.83},
407. {40.83,87.14},{13.54,35.77},{31.44,62.92},{79.69,110.30},
408. {67.07,100.59},{28.81,78.71},{52.95,97.30},{39.89,81.67},
409. {58.79,75.89},{34.35,51.29},{38.03,64.97},{87.87,130.19},
410. **{39.73,52.43},{ 1.64,31.22},{91.15,147.58},{54.08,101.10},**
411. {53.53,74.54},{54.24,104.47},{15.04,51.28},{79.06,114.59},
412. {93.83,138.37},{94.89,122.18},{52.63,86.22},{27.83,68.05},
413. {54.51,94.07},{23.83,58.00},{86.88,141.66},{10.42,31.81},
414. {55.43,84.31},{45.04,85.30},{95.69,121.78},{17.28,35.32},
415. **{ 3.17,33.76},{51.61,69.81},{27.37,64.13},{88.92,160.98},**
416. {31.40,64.46},{33.35,59.91},{82.48,128.89},{50.46,98.13},
417. {78.73,113.68},{70.08,115.27},{98.65,142.28},{ 9.15,50.95},
418. {16.74,35.73},{32.92,72.02},{ 1.29,18.94},{75.79,123.45},
419. {32.94,59.92},{61.72,81.50},{42.39,91.90},{70.15,108.81},
420. **{ 2.90,29.10},{59.68,87.41},{69.85,108.66},{71.21,107.81},**
421. {24.09,46.47},{44.51,76.59},{ 7.30,34.83},{58.93,99.24},
422. { 1.24,22.60},{84.27,132.21},{54.11,87.19},{39.18,75.93},
423. {90.81,155.72},{67.68,88.19},{67.14,84.53},{53.98,86.47},
424. {67.28,106.68},{ 8.49,36.74},{34.96,62.55},{59.01,82.94},
425. **{64.78,101.77},{66.24,110.82},{75.81,131.28},{62.82,76.02},**
426. {73.95,116.37},{20.40,38.76},{45.06,84.65},{47.64,82.81},
427. {30.85,64.41},{77.10,112.67},{ 8.12,32.76},{39.56,53.41}
428. };

**Explanation of MPI Linear Regression:**

The main motive of this code is to find out the minimum value of m, c and error. To calculate time two time-spec start and finish along with time elapsed is declared in declaration phase to access the start time, finish time and the time difference respectively. Clock\_get time function is used to get the time from the system which takes two parameters CLOCK\_MONOTONIC and &start or CLOCK\_MONOTONIC and &finish to get the start time and finish time respectively. Time elapsed is calculated by subtracting finish time and start time after getting the both the time. In MPI, we must declare to variable size and rank and MPI library is initialized by MPI\_Init datatype. MPI\_Comm\_size() function is used to identify size of communicator and MPI\_Comm\_rank() function is used to identify the rank of the calling process. We have declared the bm, bc and best\_error with values 1.3, 10 and 999999999. rms\_error() with parameter m and c is the function to find the error of the straight line. Rms error can be calculated by square rooting all sum of squares of error divided by number of errors. Mathematically, it can be represented by:

Root Mean Square error=

Residual error is calculated by the difference between observed value and predicted value. Before finding the RMS error we must find the residual error to find the minimum error.

Residual error = observed value – predicted value

Inside rms\_error() function, another function residual\_error() function with parameter double x, double y, double m, double c. The value of x and y are taken from the data provided by the university to us. The residual error is calculated by the formula e = (m \* x) + c – y and square of e is returned to the function rms\_error(). The program only executes if there are exactly nine processes. The MPI program MPI\_Send() is used to do blocking send to other rank from the current rank. MPI\_Send has six parameters:

1. address of send buffer

2. count in send buffer

3. datatype of each send buffer element

4. rank of destination

5. message tag

6. communicator

We have used MPI\_Send() to send the work to the rank 0. We used MPI\_DOUBLE as a datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator.

Similarly, MPI\_Recv() is used to receive for sent message from another rank to the current rank and it has 7 parameters. They are:

1. address of send buffer

2. Count in send buffers

3. datatype of each send buffer element

4. Rank Source of the sender

5. message tag

6. communicator

7. Status

We have used MPI\_Recv() to receive the work from rank 0. We used MPI\_INT datatype, 0 as message tag and MPI\_COMM\_WORLD as a communicator. We ignored the status of the receive with MPI\_STATUS\_IGNORE keyword.

After completion, the total time taken by the program is calculated by the function named time\_difference() function. Since we have used MPI library specification, we must use MPI\_Finalize() function to terminates the execution environment of MPI.

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

|  |  |  |
| --- | --- | --- |
| S.N. | Running time (Original Program) | Running time (MPI) |
| 1 | 52689676ns or 0.052689676s | 312739562ns or 0. 312739562s |
| 2 | 52694668ns or 0.052694668s | 286004524ns or 0. 286004524s |
| 3 | 52739608ns or 0.052739608s | 269691331ns or 0. 269691331s |
| 4 | 52767176ns or 0.052767176s | 262587604ns or 0. 262587604s |
| 5 | 52658329ns or 0.052658329s | 297469760ns or 0. 297469760s |
| 6 | 52725256ns or 0.052725256s | 268117311ns or 0. 268117311s |
| 7 | 53270258ns or 0.053270258s | 314620163ns or 0. 314620163s |
| 8 | 52675616ns or 0.052675616s | 278382238ns or 0. 278382238s |
| 9 | 53785147ns or 0.053785147s | 271421042ns or 0. 271421042s |
| 10 | 52968744ns or 0.052968744s | 296962980ns or 0. 323081038s |
|  | Mean Running time for original program: 52897447.8ns or | Mean Running Time (MPI version): 285799651.5ns or 0.2857996515s |

Table 12 comparison of running time between original program and MPI version of linear regression

**Write a short analysis of the results:**

In MPI, one process passes message to other process within the communicator. So, it is obvious that the runtime taken by the program using MPI would be much less than the original program. The Mean Running time for original program is 52897447.8ns or 0.52897447s. After running the MPI program the result of mean running time is found to be 285799651.5ns or 0.2857996515s which is much lesser than the original program. The program that use MPI is =1.85085764529 times faster than the original program. The program runs faster if the standard message passing library compared to the original program because MPI divides the work in different processes which can be identified by rank. Since, we have used rank and processes, the mean run time for this program is lesser than the original program.

# Verbose:

akp@LAPTOP-GPT9MJRS MINGW64 ~/Desktop/HPC (master)

$ git log

commit 84a6a615b4d9dca64ef2a4812652c21a69788e08 (HEAD -> master, origin/master, origin/HEAD)

Author: unknown <bishalkhadka75919@gmail.com>

Date: Thu Dec 26 10:42:39 2019 +0545

message

Git link: git clone https://bishalkhadka1@bitbucket.org/bishalkhadka1/hpc.git .