6CS005 Learning Journal - Semester 1 2019/20

Miraj Pudasaini 1928185

# Table of Contents

[Table of Contents 1](#_Toc527650969)

[1 POSIX Threads 2](#_Toc527650970)

[1.1 Password Cracking 2](#_Toc527650971)

[1.2 Image Processing 2](#_Toc527650972)

[1.3 Linear Regression 2](#_Toc527650973)

[2 CUDA 3](#_Toc527650974)

[2.1 Password Cracking 3](#_Toc527650975)

[2.2 Image Processing 3](#_Toc527650976)

[2.3 Linear Regression 3](#_Toc527650977)

[3 MPI 4](#_Toc527650978)

[3.1 Password Cracking 4](#_Toc527650979)

[3.2 Image Processing 4](#_Toc527650980)

[3.3 Linear Regression 4](#_Toc527650981)

[4 Verbose Repository Log 5](#_Toc527650982)

# POSIX Threads

## Password Cracking

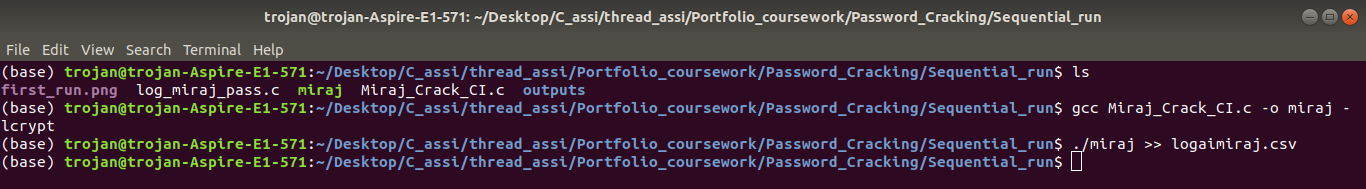


Figure Running password cracking algorithm

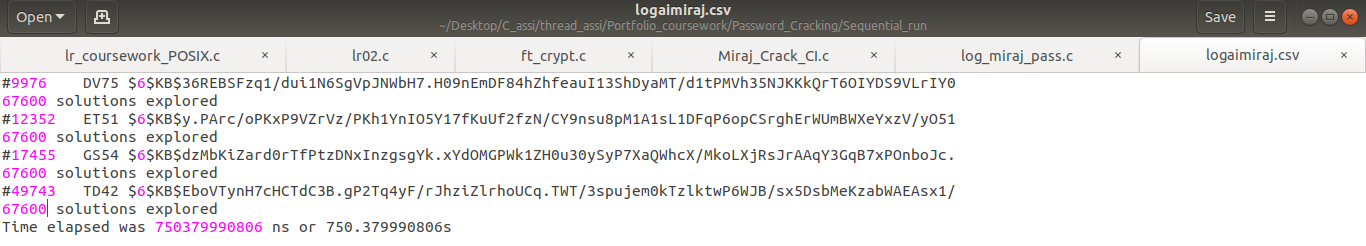
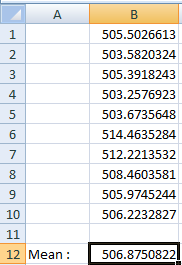


Figure Cracked passwords

**Mean running time for the sequential program.**

The program was run ten times manually and the mean running time for was calculated as follows:



The mean execution time of the algorithm to brute force the passwords of two initials and two digits came around 8.4 mins in calculations  
as there were four passwords in one single algorithm the mean execution time per one password could be estimated around 2.1 mins per password.

**Execution time as the variables like length of password increases (A Short Estimation).**

As seen on above exploration of the algorithm the mathematical expressions (using **principles of permutation and combination)** for cracking the two initial two numerical passwords is as illustrated on table one:

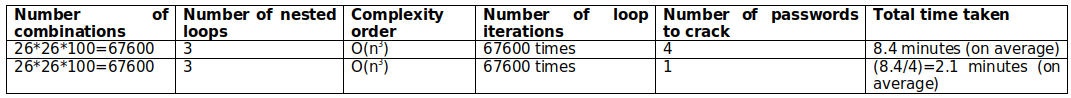
****

Figure Estimation Table one

On adding one letter to the combination the following increase of tested combination goes through the algorithm. Accordingly the **time complexity** of the algorithm increases by exponent of order of one which in compare to that of two initial.

Since,

N3 clutter of loopresulted on 2.1 minutes of execution time per password (for two initials)  
 N4 clutter of loop for three initial shall result on

Here for the increased number of alphabet result in combination nA = 67600\*26=1757600 combinations check

Execution of 1757600 iteration= 218.4minutes (for four password) = 218.4/4 = 54.6minutes for one single password

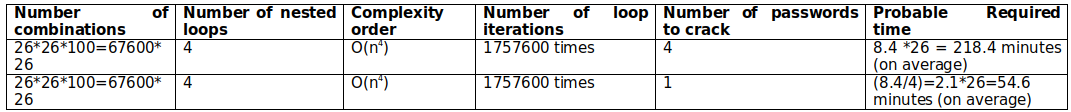


Figure Estimation of table two

Hence an estimation of **218.4 mins** is approximated for the all of four three initial password to get cracked provided with the same resources for processing speed and without adding any optimization techniques on the source code is made for answering this section.

**Note:**The mathematical explanation for the algorithm’s execution time are calculations neglecting the constants like syntax and variable memory initialization so in case of loosely handling memory management in C the execution time may vary depending upon the constants.

**Source code for three initial:**

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <crypt.h>  #include <sys/stat.h>  #include <time.h>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  this program will decrypt passwords which is combination of 3 initials and 2 digits.  an extra outer for loop has been added to iterate entire loop yet another 26 times. the loop control variable 'w'  has been used to designate the leftmost initials of the password.    a password hash has been generated using SHA512 algorithm as follows:  password : ZZZ99    hash value : $6$KB$.mzYlKZH98ERJvUQ4W2HLVk4bQuv.E/AjFR0SYIXgrutOdtW27ojLaBXhgZGvkYqA6PPEQw5y2omHyD4j1u4B/  the hash value has been assigned to the string variable called "encrypted\_password" below:    the program will print the password and hash value when match is found.  since there are 4 loops the order of complexity of the program n^4  total number of loop executions in this case will be  26\*26\*26\*100=1757600 times.    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Char \*encrypted\_password= "$6$KB$.mzYlKZH98ERJvUQ4W2HLVk4bQuv.E/AjFR0SYIXgrutOdtW27ojLaBXhgZGvkYqA6PPEQw5y2omHyD4j1u4B/";  int time\_difference(struct timespec \*start, struct timespec \*finish, long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  /\*\*  Required by lack of standard function in C.  \*/  void substr(char \*dest, char \*src, int start, int length){  memcpy(dest, src + start, length);  \*(dest + length) = '\0';  }  /\*\*  This function can crack the kind of password explained above. All combinations  that are tried are displayed and when the password is found, #, is put at the  start of the line. Note that one of the most time consuming operations that  it performs is the output of intermediate results, so performance experiments  for this kind of program should not include this. i.e. comment out the printfs.  \*/  void crack(char \*salt\_and\_encrypted){  int w,x, y, z; // Loop counters  char salt[7]; // String used in hashing the password. Need space for \0  char plain[7]; // The combination of letters currently being checked  char \*enc; // Pointer to the encrypted password  int count = 0; // The number of combinations explored so far  substr(salt, salt\_and\_encrypted, 0, 6);  for(w='A'; w<='Z'; w++){  for(x='A'; x<='Z'; x++){  for(y='A'; y<='Z'; y++){  for(z=0; z<=99; z++){  sprintf(plain, "%c%c%c%02d",w, x, y, z);  enc = (char \*) crypt(plain, salt);  count++;  if(strcmp(salt\_and\_encrypted, enc) == 0){  printf("#%-8d%s %s\n", count, plain, enc);  } else {  //printf(" %-8d%s %s\n", count, plain, enc);  }  }  }  }  }  printf("%d solutions explored\n", count);  }  int main(int argc, char \*argv[]){  struct timespec start, finish;  long long int time\_elapsed;  int i;  clock\_gettime(CLOCK\_MONOTONIC, &start);  crack(encrypted\_password);    clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &time\_elapsed);  printf("Time elapsed was %lld ns or %0.9lfs\n", time\_elapsed,  (time\_elapsed/1.0e9));  return 0;  } |

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

**Analysis of Estimated Time vs Actual Time**



The code inside the pccw\_3initials.c was compiled and ran on the PC made available at the college lab since the resource on my personal PC were not enough execute the algorithm with such time complexity.



This was the output as observed from the execution. To compare; the program for brute forcing three initial took approximately **344.719884220s** in conversion can be said that it took 55.4 minutes. Comparing it to the estimated time.   
 Te=54.6minutes  
 (Real observed time) Tr=55.4minutes   
Considering the extra execution time as change in execution time from mathematical explanation as del(time).

Del(time)= Tr-Te = 55.4 -54.6=0.8minutes

A small amount of del(time) can be explained with respect to the constants discussed earlier since we didn’t included the memory processing of them in the estimation. Theoretically speaking we didn’t made estimation of execution of all the flushing and reinitiating of the loop variables however it still do not explain the rate of change ‘del(time)’ being 3%.   
For explanatory reasoning’s I have hypothesised that this has happened due to the shared memory architecture on the core processors. At the instance while running this algorithm the CPU scheduling for ruining the batch job of this particular execution might have gone through various state transfers inside the CPU since there are almost 100’s of kernel process instances running on the same shared memory architecture which made this batch job of this process to halt and execute several time which might have created an exponential chain effect to occur such delay.

**Multithreaded Version of two initials (Using posix and optimized Mutex for posix threads)**

For the purpose of using clutter free loops and fetching the thread with desired chunk of and for **load balancing** between two threads a simple data optimization technique has been used which is called **Domain Decomposition.** The domain decomposed divides the data into desirable chunk.



Here the 67600 combination is divided into two equal chunk as Domain Decomposition.

**Mutex for thread synchronization**

Mutex are inbuilt kernel functions provided on UNIX based operating Systems to restrict the posix thread from collusion while being executed on the same CPU. Although POSIX threads are an exercise of parallel processing but if we examined that with respect to processor execution of the whole algorithm batch job we will observe that the process executes each ready stated execution one at a time since both the thread will have origin (PID) of same execution cycle the CPU will not recognize our threads as two different process this technique of parallel process of data to compiler before execution is called **User Level Threading**. Hence in order for both threads to work simultaneously without being confused with each other and wasting time to decide which process should execute first we use a Scheduling and Exclusion technique called MUTEX (Mutual Exclusion).

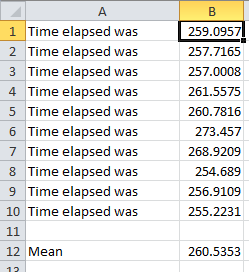
There are other such techniques like ***Forking, Using Semaphores on Mutex, Using Futex*** instead however we only deal with user level parallelism in this part since we are using CUDA for further analysis and demonstration of pseudo parallelism.  
General Principle for mutex to work is the use of MUTEX locking variables which is demonstrated as follows:

Source code:

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <pthread.h>  #include <crypt.h>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Compile and run with:  cc -o pccw\_multithreadVersion pccw\_multithreadVersion.c -pthread -lcrypt  ./pccw\_multithreadVersion    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* a mutex lock has been defined with global scope so that it could be used  by both of the threads \*/  pthread\_mutex\_t mutex\_lock;  int n\_passwords = 4;  /\* variable for counting solutions examined by two threads \*/  int solution\_counter;      /\* those variables are made global so that each thread can access them \*/  char \*saltAndEncryptedText;  char salt[7];  char \*enc;  /\*\*\*\*\* 4 hash value of password supplied \*\*\*\*\*\*\*/  char \*encrypted\_passwords[] = {  "$6$KB$36REBSFzq1/dui1N6SgVpJNWbH7.H09nEmDF84hZhfeauI13ShDyaMT/d1tPMVh35NJKKkQrT6OIYDS9VLrIY0",  "$6$KB$y.PArc/oPKxP9VZrVz/PKh1YnIO5Y17fKuUf2fzN/CY9nsu8pM1A1sL1DFqP6opCSrghErWUmBWXeYxzV/yO51/",  "$6$KB$dzMbKiZard0rTfPtzDNxInzgsgYk.xYdOMGPWk1ZH0u30ySyP7XaQWhcX/MkoLXjRsJrAAqY3GqB7xPOnboJc.",  "$6$KB$EboVTynH7cHCTdC3B.gP2Tq4yF/rJhziZlrhoUCq.TWT/3spujem0kTzlktwP6WJB/sx5DsbMeKzabWAEAsx1/"  };  int time\_difference(struct timespec \*start,  struct timespec \*finish,  long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  /\* A function to extract substring from a src string to dest string starting  from index 'start' upto index 'length' \*/  void substr(char \*dest, char \*src, int start, int length){  memcpy(dest, src + start, length);  \*(dest + length) = '\0';  }  /\* the thread function to decrypt hash and compared it against the hash value supplied  the function makes comparsion brute-forcely. the function check for password starting form initial  letter 'A' to 'M' \*/  void \*crackA2M(){  int x, y, z; // Loop counters  char plain[7];  for(x='A'; x<='M'; x++){  for(y='A'; y<='Z'; y++){  for(z=0; z<=99; z++){  sprintf(plain, "%c%c%02d", x, y, z);  enc = (char \*) crypt(plain, salt);  /\* locking critical section of the program \*/  pthread\_mutex\_lock(&mutex\_lock);    solution\_counter++;    /\* releasing lock from critical section of the program \*/  pthread\_mutex\_unlock(&mutex\_lock);    if(strcmp(saltAndEncryptedText, enc) == 0){  printf("#%-8d%s %s (computed by thread 1 : ID %ld)\n", solution\_counter, plain, enc,pthread\_self());  } else {  //printf("%-8d%s %s (computed by thread %ld)\n", count, plain, enc,pthread\_self());  }  }  }  }  pthread\_exit(NULL);  }  /\*\*\* the purpose of this function is same as that of 'crackA2M' except that it checks for passwords  starting from 'N' to 'Z' \*\*/  void \*crackN2Z(){  int x, y, z; // Loop counters  char plain[7];  for(x='N'; x<='Z'; x++){  for(y='A'; y<='Z'; y++){  for(z=0; z<=99; z++){  sprintf(plain, "%c%c%02d", x, y, z);  enc = (char \*) crypt(plain, salt);    /\* locking critical section of the program \*/  pthread\_mutex\_lock(&mutex\_lock);    solution\_counter++;    /\* releasing lock from critical section of the program \*/  pthread\_mutex\_unlock(&mutex\_lock);    if(strcmp(saltAndEncryptedText, enc) == 0){  printf("#%-8d%s %s (computed by thread 2 : %ld)\n", solution\_counter, plain, enc,pthread\_self());  } else {  //printf("%-8d%s %s (computed by thread %ld)\n", count, plain, enc,pthread\_self());  }  }  }  }  pthread\_exit(NULL);  }  /\* a function to implement 2 threads \*/  void crack(char \*salt\_and\_encrypted){  solution\_counter=0;  saltAndEncryptedText=salt\_and\_encrypted;  pthread\_t t1, t2;  substr(salt, salt\_and\_encrypted, 0, 6);  if(pthread\_create(&t1, NULL, crackA2M, NULL)!=0){  printf("Sorry, Thread could not be created !\n");  exit(EXIT\_FAILURE);  }    if(pthread\_create(&t2, NULL, crackN2Z, NULL)!=0){  printf("Sorry, Thread could not be created !\n");  exit(EXIT\_FAILURE);  }  pthread\_join(t1, NULL);  pthread\_join(t2, NULL);  printf("%d solutions explored\n",solution\_counter); //shows total number of comparsion performed  }    int main(int argc, char \*argv[]){    struct timespec start, finish;  long long int time\_elapsed;  int i;  clock\_gettime(CLOCK\_MONOTONIC, &start);  /\*\* initialization of mutex lock with error checking \*\*/  if(pthread\_mutex\_init(&mutex\_lock, NULL)!=0) {  printf("problem with initialising mutex...\nProgram terminating...");  exit(EXIT\_FAILURE);  }  for(i=0;i<n\_passwords;i<i++) {  crack(encrypted\_passwords[i]);  }  /\* destroying of the mutex lock at the end \*/  pthread\_mutex\_destroy(&mutex\_lock);  clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &time\_elapsed);  printf("Time elapsed was %lld ns or %0.9lfs\n\n", time\_elapsed,  (time\_elapsed/1.0e9));  return 0;  } |

**Analysis of mean execution time for Sequential Run and Parallel Runs**

After running the program times with mr.py python script and exporting the results in a .csv file the following mean time was calculated.



For the rate of execution (i.e harmonic mean)

We have,

n / ∑ [1/Xi]

= 260.53503214584seonds

Since the MRTs= 260.5353 second per program

Here as observed earlier the sequential program which took 8.3minutes running in single process in compare to this version with two thread enabled took approximately half of the time taken by the sequential version for execution of the same algorithm and dataset. This is an illustration of use of threads to optimize the work load on a process and a proof that using thread divides the process as desired.

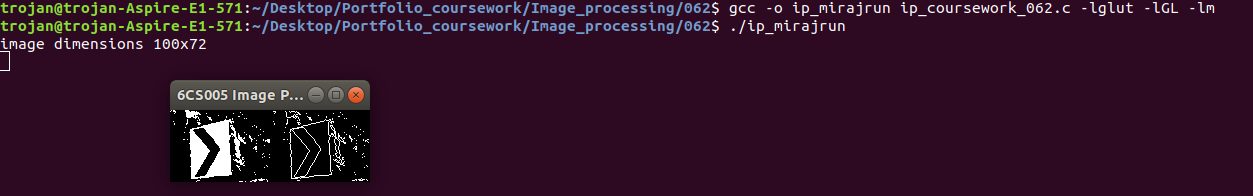
## Image Processing

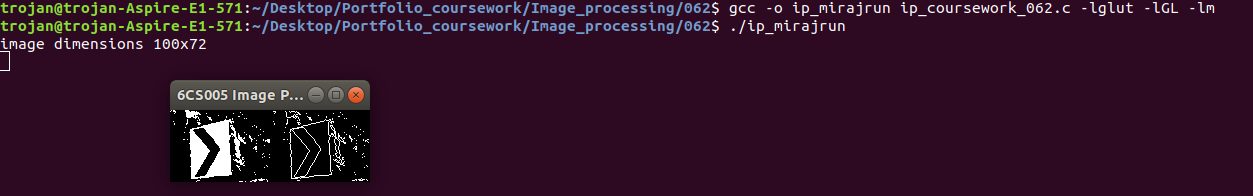
a)



This is the 100\*72 image provided in the 062 course work.

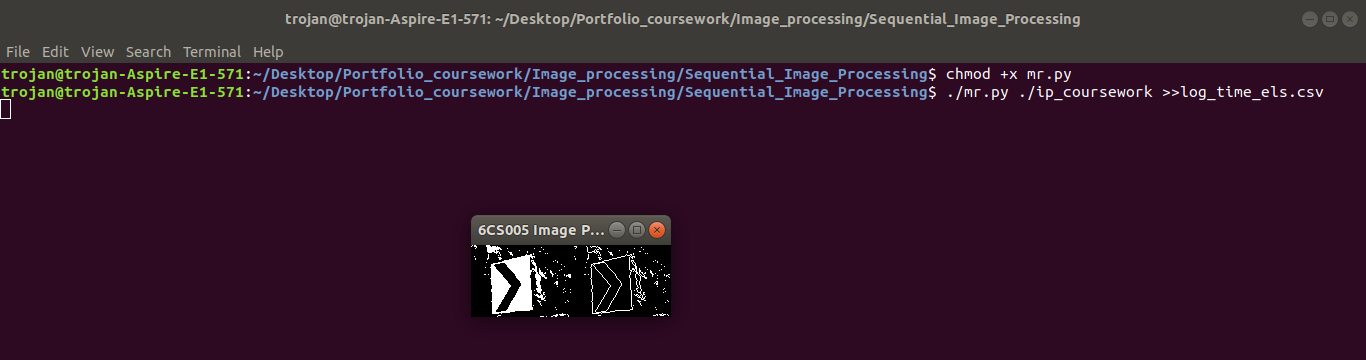
On the first simple manual run of the program provided on a linux bash terminal following output was observed



  
Illustration 1: Simple run of the provided program

By running the **sequential version** of the pogram for 10 times with a python script the mean running time was calculated.

For calculation of mean running time weighted measure performance measuring technique was followed which furhter more detail can be found here.([)](http://ece-research.unm.edu/jimp/611/slides/chap1_2.html)

  
Illustration 2: Sequential Program 10 runs

The mean running time was calculated from the *log\_time\_els.csv* file:

|  |  |
| --- | --- |
| **Iteration** | **Time Elapsed** |
| First Iteration | 128052ns/0.000128052s |
| Second Iteration | 129700ns/0.000129700s |
| Thrid Iteration | 140614ns/0.000140614s |
| Fourth Iteration | 195262ns/0.000195262s |
| Fifth Iteration | 124384ns/0.000124384s |
| Sixth Iteration | 124384ns/0.000124384s |
| Seventh Iteration | 86716ns/0.000086716s |
| Eight Iteration | 209098ns/0.000209098s |
| Ninth Iteration | 126886ns/0.000126886s |
| Final Iteration | 116201ns/0.000116201s |
| **Mean Running Time** | 0.000138324s |

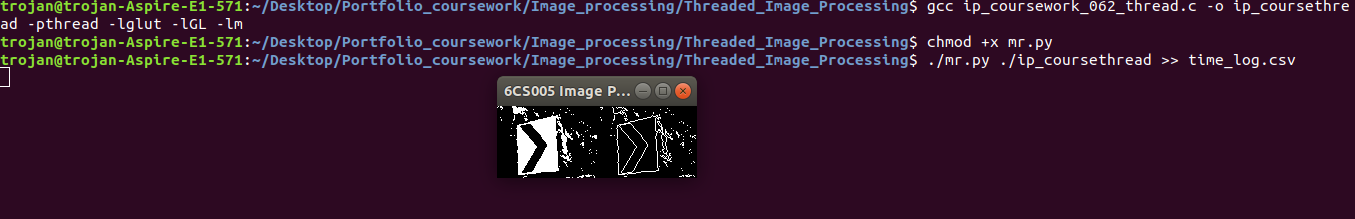
b)

Source code for image processing with multithreading with POSIX thread.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <GL/glut.h>  #include <GL/gl.h>  #include <malloc.h>  #include <signal.h>  #include <pthread.h>  #include <unistd.h>  #define widthx 100  #define heightx 72  #define n 4  #define total\_pixel widthx\*heightx  //struct for thread strides  typedef struct {  int start\_index;  int stride;  } t\_args;  //inbuilt structure for time calculation  struct timespec start, finish;  // variable to hold time difference  long long int time\_el;  unsigned char image[], results[widthx \* heightx];  //since 4 threads will be used, 4 different thread\_Argument variables are required  t\_args targ\_array[n];  int time\_difference(struct timespec \*start, struct timespec \*finish,  long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  //thread function that will be called by 4 threads which will detect edges  void \*detect\_edge\_thread\_function(void \*ptr1){    t\_args \*ptr=ptr1;  for(int i=ptr->start\_index;i<total\_pixel;i=i+ptr->stride){    int x, y; // the pixel of interest  int b, d, f, h; // the pixels adjacent to x,y used for the calculation  int r; // the result of calculate    y = i / widthx;  x = i - (widthx \* y);  if (x == 0 || y == 0 || x == widthx - 1 || y == heightx - 1) {  results[i] = 0;  } else {  b = i + widthx;  d = i - 1;  f = i + 1;  h = i - widthx;  r = (image[i] \* 4) + (image[b] \* -1) + (image[d] \* -1) + (image[f] \* -1)  + (image[h] \* -1);  if (r > 0) { // if the result is positive this is an edge pixel  results[i] = 255;  } else {  results[i] = 0;  }  }      }  //ensuring that actualy 4 threads have been created by the program  printf("Thread ID: %ld\n",pthread\_self());  pthread\_exit(NULL);    }  void detect\_edges() {  pthread\_t thread\_array[n];    //reading of time right before starting the edge detecting process begins  clock\_gettime(CLOCK\_MONOTONIC, &start);    //loop for initializing the thread\_Argument struct variables and creating 4 threads  for(int i=0;i<n;i++){  targ\_array[i].start\_index=i;  targ\_array[i].stride=4;  pthread\_create(&thread\_array[i], NULL, detect\_edge\_thread\_function, &targ\_array[i]);  }  //waiting for 4 different threads to finish their job and get joined back to the main thread  for(int i=0;i<n;i++){  pthread\_join(thread\_array[i], NULL);  }  //reading of time right after finishing the edge detecting process  clock\_gettime(CLOCK\_MONOTONIC, &finish);  //calculating time difference  time\_difference(&start, &finish, &time\_el);  printf("Time elapsed was %lldns or %0.9lfs\n", time\_el,  (time\_el/1.0e9));  }  //parameters for the display window using free glute library called from main function  static void display() {  glClear(GL\_COLOR\_BUFFER\_BIT);  glRasterPos4i(-1, -1, 0, 1);  glDrawPixels(widthx, heightx, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);  glRasterPos4i(0, -1, 0, 1);  glDrawPixels(widthx, heightx, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);  glFlush();  }  //function to terminate the execution when the user presses the key in the interuption handling function  void tidy\_and\_exit() {  exit(0);  }  //Handling key press  void sigint\_callback(int signal\_number){  printf("\nInterrupt from keyboard\n");  tidy\_and\_exit();  }  //handling user interuption  static void key\_pressed(unsigned char key, int x, int y) {  switch(key){  case 27: // escape  tidy\_and\_exit();  break;  default:  printf("\nPress escape to exit\n");  break;  }  }  //main function  int main(int argc, char \*\*argv) {  signal(SIGINT, sigint\_callback);    printf("image dimensions %dx%d\n", widthx, heightx);  detect\_edges();  glutInit(&argc, argv);  glutInitWindowSize(widthx \* 2,heightx);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);    glutCreateWindow("6CS005 Image Progessing Courework Multithreaded version");  glutDisplayFunc(display);  glutKeyboardFunc(key\_pressed);  glClearColor(0.0, 1.0, 0.0, 1.0);  glutMainLoop();  tidy\_and\_exit();    return 0;  }  //array matrix of imgae provided  unsigned char image[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,  255,0,0,0,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,0,  255,255,255,255,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,0,255,255,255,255,255,255,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,  255,255,255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,255,255,0,0,0,255,0,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,  255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,255,255,255,255,255,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,  0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,  255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,  0,0,255,0,0,0,0,255,0,0,0,0,0,0,255,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,255,  255,255,255,0,255,255,0,0,0,0,0,255,255,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,255,255,0,255,255,255,255,0,0,255,0,0,0,0,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,0,0,  255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  0,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,255,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,  0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,255,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  0,0,255,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,255,255,255,255,0,255,0,0,0,0,0,0,0,0,0,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  }; |

After running the previous version the sequential version of the program was modified to sub divide the process batch job for parallel processing and hence 8 equal **POSIX Thread** executed the sequential program, with the help of the *‘mr.py’* script this version was also ran for 10 times:

After analysing the results from the **‘***time\_log.csv****’*** *a mean running time was analysed as follow:*

  
Illustration 3: Threaded Version 10 run

|  |  |
| --- | --- |
| **Iteration** | **Time Elapsed** |
| First Iteration | 711637ns or 0.000711637s |
| Second Iteration | 2174209ns or 0.002174209s |
| Thrid Iteration | 677832ns or 0.000677832s |
| Fourth Iteration | 559941ns or 0.000559941s |
| Fifth Iteration | 559941ns or 0.000559941s |
| Sixth Iteration | 748271ns or 0.000748271s |
| Seventh Iteration | 894318ns or 0.000894318s |
| Eight Iteration | 510498ns or 0.000510498s |
| Ninth Iteration | 748473ns or 0.000748473s |
| Final Iteration | 517468ns or 0.000517468s |
| **Mean Running Time** | 754264.7ns or 0.000754265s |

*c)*

On observation,

M.R.T(mean running time) for the Sequential Version of the program MRTs= 0.000138324s

MRTips Harmonic(rate of execution per program)= n / ∑ [1/Xi]= 0.000136542s

M.R.T(mean runnig time) for the Multi Threaded Parallel processed program MRTp=0.000754265s

MRTipp Harmonic(rate of execution per program)= n / ∑ [1/Xi]= 0.000136542s

Since the MRTp > MRTs

which means that on a batch execution with no pseudo parallelism (I.e on a single CPU) the execution time for the Multi Threaded program which used four thread execution took longer to execute than the normal one hence the use of parallelism seems to be unecessary for the execution time comparision. Instead of this, **Forking** the batch job(main process) into smaller processes or Using **Semaphores** might have helped on improving the optimization . *The reason for the threads to not reduce the execution time is estimated to be the small data* **throughput** *that was fetched*. It can be explained as if the time for creation and the exectution of the threads were larger than the exection of the edge detection algorithm.

Paste your source code for your multithread password cracker here

On a process level with a normal clocking speed CPU (3.5GHz) ignoring the *Wallclock time and Latency* of the CPU:

The process time for batch job execution of the threads alone can be calculated as

MRTt(Mean Run Time for Threads) = MRTp – MRTs= 0.000138324-0.000754265=0.000615941s

Here we can see that MRTt is highly greater than MRTs which means that creating and executing the threads were more costly than executing the edge detection sequentially hence it is discovered that for small through put use of multi threading could result in more costly executiong time hence proven more time consuming.

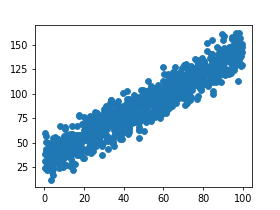
From the first table Sequential run for 10 time, if the data were to be plotted with X ordinate as ith iteration and Y absisa as the execution time it seems to follow a **sinosidual curve(sine curve)** (I.e the execution time of first few iteration were decreasing and suddenly from fourth iteration the execution time increased and then again decreased for two iterations reachead at a peak and finally on the final execution decreased to where it started which if plotted as a graph would result into a wave like structure).

This can be described as if the CPU on first iteration of execution had no blocked or ready execution on the batch job index hence with max clocking executed the program but following the piling up of the other iteration the clock speed might have been distributed and it took a bit longer to execute the algorithm following this cycle the formation of sinosidual curve.

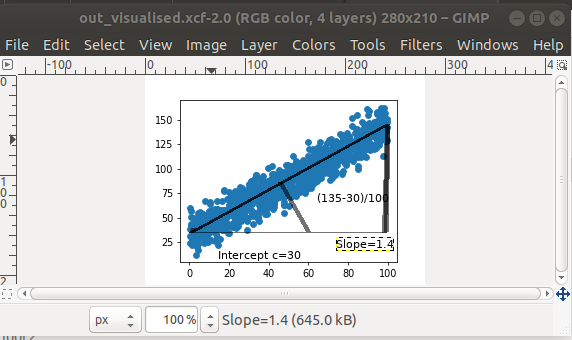
From the second table User level prallelism with POSIX’s 10 iteration, if the data were plotted in the graph it would follow a **Cosine(Co-sinosidual) curve (cose curve)** which means that the execution time were higher and then lower and were fluctuating heavily following the cosine curve cycle. Which is quite unusual and opposite comparing it to the sequential version.   
 This could be described as the CPU architecture for handeling the workload latency and creation and execution of kernel level thread space and thread memory initialization.

## Linear Regression

a) The data provided from the university was fetched with the source code of lr00.c and the absicass and y intercept were plotted with the help of numpy and matlibplot (python libraries) the source code of which is included on the py.py python file included on the zip protfolio.

  
Illustration 4: Scatter Plot of the given ordinate and y intersection

b)

  
Illustration 5: Slope and y intercept visualization

The above scattered plot visualized and from the figure above we can observe that   
 y-intercept is about 30 units and the slope of the visualization is about 1.4 units.

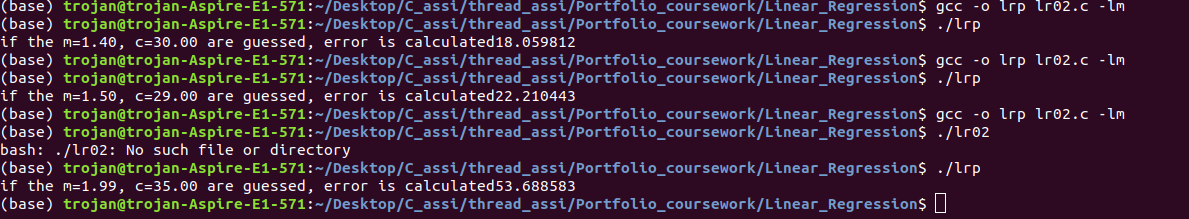
Here for optimium guess of m and c three times are as follows

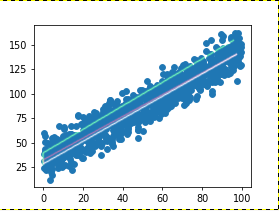
**M1 = 1.3 and C = 30**

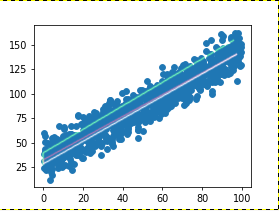
**M2=1.5 and C=29**

**M3=1.99 and C=35**

For another iteration the dataset provided was ran with the source code for finding the rms mean slope and c the hard coded values for m and C were replace and ran for three times for the three guesses individually. The output observed are as follows:

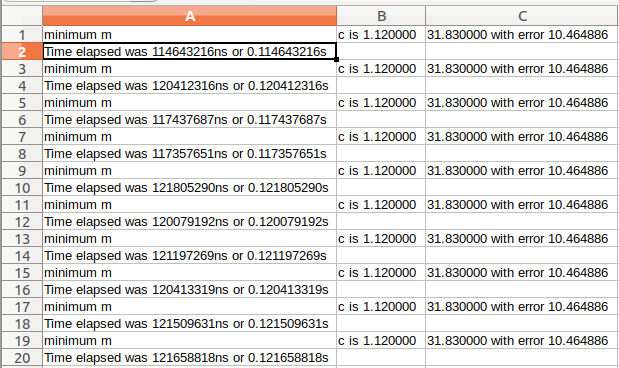
  
Illustration 6: Slope C and RMS for consiquitive guess

Illustration 7: A graph overlay of the guesses

  
Illustration 8: Observation of regression lines of M and C with the result

Here on the illustration of the line formed

as seen over the scatter plot the first guess gave the **red regression line** the second guess gave the **white regression line** and the sky blue **regression line** was as plotted by the third guess. As it can be seen that the ***rms error*** for the first run was 18.059812 the regression line was less disperresed the same was the case for the second regression line. However for the third guess the **rms error** was 53.688583 which led the regression line to be more inclined towards the outbound of the scatter plot. The scatter plot here shows a **higher corelation of linearity (high linear corelation).**



Here is table for ten times run of the sequential program along with the M and C.   
Mean running time for the program as calculated is:

MRTlrs= **∑ xi/ n**

=0.119651439 second is the mean run time   
since mean runtime needs to be calculated as rate the harmonic mean run time would be:

MRTlrs Harmonic= n / ∑ [1/Xi]

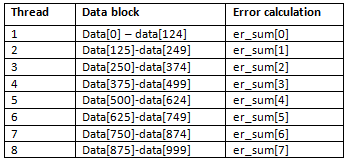
hence,

del(execution time)= 0.119607871 per program.

Running Linear Regression Algorithm with POSIX thread

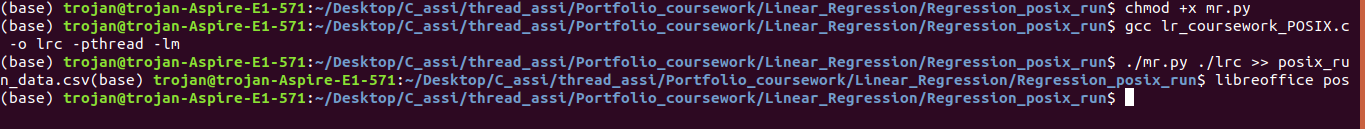
For applying the linear Regression the part in which the algorithm finds the residual error was divided into chunk and was divided into threads for parallel execution. Following the similar techniques used in password cracking for ease execution and data serialization Domain Decomposition technique for data was applied.

Considerate the huge scale of coordinates (data) being fetched into the algorithm I have decided to equi-partation the data among threads and all of the threads will be fetched equal amount of data for load balancing among them.   
On analysis the RMS in the data = 1000   
Hence for each thread the execution chunck would be= 1000/8=125 data points per thread.  
As illustrated Domain Decomposition with respect to Error calculation for each thread (table below):



Source code for POSIX based linear Regression :

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <pthread.h>  #include<time.h>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* To compile:  \* cc -o lr\_coursework\_POSIX lr\_coursework\_POSIX.c -lm -pthread  \*  \* To run:  \* ./lr\_coursework\_POSIX  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int time\_difference(struct timespec \*start, struct timespec \*finish,  long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  //struct definition for point (x,y)  typedef struct {  double x;  double y;  } point\_t;  //struct definition for passing as argument to the tread function  //variable startingIndex serves for initialization expression in FOR loop inside thread function  //variable errorSumIndex serves for ideintifying error\_sum for a particular thread computation  struct blocksize{  int startingIndex;  int errorSumIndex;  };  //since 8 threads are used there needs to be 8 different instances of blocksize struct data type  struct blocksize block[8];  int n\_data = 1000;  //since slope and intercept are same for each instance of thread while computing residual error these  //are declared as global variable  double slope, intercept;  //each 8 thread computes residual\_error for 125 data points therefore array of 8 er\_sum is required  //to hold sum of error computed by 8 diferent threads  double er\_sum[8];  point\_t data[];  double residual\_error(double x, double y) {  double e = (slope \* x) + intercept - y;  return e \* e;  }  //sequential looping portion from lr05.c program has been taken out and placed inside thread function  //called thread\_function  void \* thread\_function(void \*ptr){  struct blocksize \*bsize=(struct blocksize \*)ptr;    //initialization of er\_sum to 0 to ensure that it doesnt hold garbage  er\_sum[bsize->errorSumIndex]=0.00;    //since 8 thread will called this function each thread will process 1000/8=125 set of points  for(int i=bsize->startingIndex; i<((bsize->startingIndex)+125); i++) {    er\_sum[bsize->errorSumIndex] += residual\_error(data[i].x, data[i].y);    }  pthread\_exit(NULL);    }  double rms\_error(double m, double c) {  slope=m;  intercept=c;  pthread\_t t[8];  double mean;  double total\_error\_sum = 0.00;  int x=0;  for (int l=0;l<8;l++){    block[l].errorSumIndex=l;    //updating starting index for each thread before being called  block[l].startingIndex=x;    pthread\_create(&t[l],NULL,thread\_function,&block[l]);    x+=125;  }    for (int l=0;l<8;l++){  pthread\_join(t[l],NULL);    //ensuring that thread are joined and summing individual sum computed by 8 different threads  //into total\_error\_sum  total\_error\_sum+=er\_sum[l];    }    mean = total\_error\_sum / n\_data;    return sqrt(mean);  }  int main() {  int i;  double bm = 1.3;  double bc = 10;  double be;  double dm[8];  double dc[8];  double e[8];  double step = 0.01;  double best\_error = 999999999;  int best\_error\_i;  int minimum\_found = 0;  struct timespec start, finish;  long long int time\_elapsed;    double om[] = {0,1,1, 1, 0,-1,-1,-1};  double oc[] = {1,1,0,-1,-1,-1, 0, 1};  clock\_gettime(CLOCK\_MONOTONIC, &start);  be = rms\_error(bm, bc);  printf("value of BE is %lf\n",be);  while(!minimum\_found) {  for(i=0;i<8;i++) {  dm[i] = bm + (om[i] \* step);  dc[i] = bc + (oc[i] \* step);  }    for(i=0;i<8;i++) {  e[i] = rms\_error(dm[i], dc[i]);  if(e[i] < best\_error) {  best\_error = e[i];  best\_error\_i = i;  }  }  // printf("best m,c is %lf,%lf with error %lf in direction %d\n",  // dm[best\_error\_i], dc[best\_error\_i], best\_error, best\_error\_i);  if(best\_error < be) {  be = best\_error;  bm = dm[best\_error\_i];  bc = dc[best\_error\_i];  } else {  minimum\_found = 1;  }  }  printf("minimum m,c is %lf,%lf with error %lf\n", bm, bc, be);  clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &time\_elapsed);  printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,  (time\_elapsed/1.0e9));  return 0;  }  point\_t data[] = {  {68.22,110.33},{65.58,111.32},{86.96,113.73},{73.01,98.54},  {68.13,106.32},{68.93,111.92},{65.44,97.07},{86.66,111.66},  {73.86,121.37},{69.19,104.82},{83.00,122.53},{47.20,79.06},  {71.74,92.95},{42.99,71.88},{10.68,55.84},{26.18,78.34},  {68.32,104.47},{50.46,79.98},{84.29,130.42},{87.36,132.92},  {72.23,98.10},{35.01,75.31},{20.48,66.49},{73.75,119.92},  {83.13,110.79},{33.87,75.12},{84.47,123.03},{63.79,116.94},  {59.77,113.15},{64.55,103.60},{63.88,111.04},{12.45,49.52},  {84.86,111.60},{39.72,100.27},{12.80,58.89},{48.43,96.61},  {40.25,80.57},{61.20,111.82},{ 6.70,41.33},{63.61,107.84},  {94.26,132.72},{57.15,108.79},{40.29,77.19},{83.69,121.57},  {48.44,91.82},{ 8.14,27.77},{10.33,50.99},{83.52,127.74},  {26.20,73.67},{83.29,136.82},{85.82,122.95},{20.20,54.89},  {26.62,71.13},{55.69,105.39},{ 8.05,67.27},{31.25,80.48},  {11.59,57.40},{20.57,62.37},{41.10,89.27},{99.26,127.74},  {55.56,96.25},{26.38,49.02},{44.80,89.43},{95.39,124.22},  {15.03,47.70},{55.63,78.89},{75.55,109.10},{80.13,121.45},  {52.49,113.23},{41.56,94.82},{55.79,76.34},{88.79,125.43},  {95.91,147.68},{ 0.78,50.39},{14.03,52.80},{67.99,118.29},  {26.40,50.08},{70.78,106.47},{64.28,87.21},{65.42,95.45},  { 2.48,48.56},{41.90,88.22},{82.38,113.24},{ 2.60,49.56},  {45.89,90.29},{31.98,52.23},{37.42,68.48},{75.27,123.44},  {63.11,115.06},{40.88,86.40},{42.37,79.09},{ 7.64,27.03},  {82.78,120.98},{63.46,107.85},{41.35,69.60},{15.17,44.93},  {48.67,78.76},{12.44,42.12},{85.48,117.51},{ 2.01,24.05},  {57.07,95.05},{ 6.70,50.66},{10.02,47.43},{74.03,120.55},  {65.64,121.43},{78.33,128.51},{66.74,115.09},{63.51,86.90},  {14.06,47.74},{47.79,62.89},{48.31,78.25},{50.04,86.84},  {59.32,95.53},{52.76,91.76},{67.79,95.90},{16.12,41.86},  { 4.39,35.29},{ 3.50,38.80},{33.57,90.90},{71.22,113.98},  {22.59,54.92},{12.11,40.82},{23.69,42.97},{17.49,49.82},  {36.60,68.93},{87.64,140.40},{42.00,59.39},{ 0.84,30.62},  { 0.82,43.22},{85.05,131.05},{17.56,37.74},{39.82,87.23},  {44.23,77.58},{50.04,61.85},{29.83,77.56},{62.06,107.86},  {47.71,73.36},{22.61,54.21},{72.28,110.52},{65.74,126.70},  {19.66,34.32},{44.45,81.12},{19.71,69.47},{ 0.89,40.05},  { 2.59,25.56},{60.26,119.54},{43.21,71.22},{51.80,93.05},  {84.07,117.57},{99.51,128.70},{65.48,109.43},{47.92,66.55},  {13.16,41.09},{63.20,95.96},{53.36,90.74},{72.61,108.64},  {93.39,135.06},{72.28,103.75},{51.74,86.51},{20.17,43.10},  {17.29,76.98},{43.98,69.98},{46.88,79.71},{46.43,87.55},  {22.02,44.51},{12.22,51.93},{68.07,114.65},{72.51,105.26},  {55.24,100.15},{49.60,96.44},{ 8.85,42.80},{29.04,69.59},  {37.25,60.77},{59.77,106.46},{24.47,47.27},{ 4.19,54.70},  {73.11,101.90},{64.35,101.97},{ 1.16,22.03},{98.30,151.28},  { 2.46,32.19},{ 6.32,31.21},{95.07,127.76},{ 1.83,25.64},  {13.93,57.57},{38.55,74.35},{98.37,140.63},{61.65,105.16},  {32.86,70.35},{78.45,105.53},{ 0.27,38.51},{52.90,93.27},  {87.87,145.93},{71.63,112.91},{ 0.78,56.79},{12.77,26.28},  {80.88,125.77},{93.37,133.81},{16.20,44.52},{75.86,122.52},  {15.69,48.28},{62.50,84.08},{55.73,78.19},{53.36,95.05},  {41.92,61.77},{69.75,119.25},{86.58,122.96},{55.17,93.91},  {95.68,130.35},{48.75,104.15},{21.67,54.50},{64.03,111.52},  {61.86,112.68},{62.11,119.55},{21.60,54.23},{56.84,81.50},  {64.28,90.37},{51.18,102.51},{37.03,67.30},{73.72,105.47},  {86.94,129.23},{36.37,67.61},{65.24,106.87},{74.61,104.20},  {37.01,58.16},{41.52,90.29},{64.62,122.13},{14.71,58.39},  {31.53,65.62},{15.13,57.93},{49.64,103.66},{69.85,107.86},  {80.81,120.29},{70.19,96.76},{86.93,133.65},{80.48,129.29},  {44.74,74.59},{80.44,120.21},{38.97,53.47},{57.04,96.35},  {72.59,107.42},{24.14,51.71},{97.64,112.78},{62.07,106.71},  {97.39,158.00},{ 7.00,54.02},{47.48,71.04},{ 6.11,54.42},  {73.69,112.84},{ 9.11,49.85},{53.86,104.77},{96.09,132.19},  {89.40,120.09},{40.97,70.15},{89.15,120.74},{32.12,81.74},  {76.85,112.80},{ 6.93,34.12},{89.23,112.16},{76.30,116.17},  {67.01,111.78},{79.65,132.04},{38.26,76.72},{ 4.92,48.89},  {90.47,135.34},{71.11,100.68},{81.14,119.41},{ 9.71,54.86},  {23.60,70.68},{ 5.89,37.04},{17.41,58.07},{58.12,75.49},  {32.07,67.72},{92.93,147.45},{43.94,85.71},{78.11,113.14},  {97.26,144.74},{83.86,128.99},{20.18,76.71},{83.84,135.20},  {48.44,88.43},{43.55,73.52},{85.48,134.95},{71.57,103.36},  {26.97,59.35},{53.70,107.61},{41.45,70.98},{ 8.94,33.27},  {19.78,62.99},{45.66,71.47},{68.49,107.93},{42.71,83.67},  {15.94,54.06},{39.90,81.69},{48.92,73.12},{27.38,76.33},  {74.12,96.24},{90.61,158.99},{44.83,74.81},{33.10,76.52},  {19.79,69.29},{71.80,115.28},{74.25,110.34},{40.21,76.62},  {62.89,105.08},{51.14,81.19},{80.93,110.24},{12.74,36.35},  {11.57,39.01},{96.31,146.26},{37.31,92.17},{76.24,118.94},  {80.55,132.96},{18.21,50.62},{40.54,79.42},{70.94,88.70},  {14.16,44.24},{59.70,85.31},{73.47,116.78},{12.70,57.68},  {99.56,147.70},{43.16,86.55},{85.46,124.35},{45.12,85.96},  {84.27,131.81},{40.10,71.28},{59.29,97.75},{24.78,63.44},  {34.27,71.82},{30.37,79.56},{ 8.66,43.75},{48.58,89.43},  {10.95,40.31},{35.27,57.13},{60.14,92.34},{35.90,63.93},  {99.18,142.25},{39.23,75.12},{78.01,123.07},{ 9.52,34.52},  {88.98,134.53},{45.28,73.47},{ 4.50,42.69},{66.42,90.37},  {94.90,139.15},{35.25,63.35},{94.76,148.77},{55.45,96.32},  {53.89,92.74},{76.85,94.58},{86.04,126.28},{ 9.60,49.99},  {77.42,130.03},{47.60,88.35},{93.65,134.52},{55.32,104.12},  {77.29,118.92},{56.72,98.08},{24.49,74.54},{21.94,56.13},  { 2.39,41.84},{57.64,83.86},{46.35,74.03},{10.92,44.50},  {42.83,73.23},{42.88,61.18},{45.32,74.45},{95.86,140.50},  {64.27,105.84},{11.12,27.35},{23.94,58.12},{14.17,51.02},  { 1.60,28.58},{10.58,48.02},{17.59,78.11},{ 5.19,56.28},  {31.24,58.15},{85.52,122.72},{39.23,78.35},{66.83,99.08},  {49.31,87.52},{83.15,123.83},{74.05,115.08},{60.07,80.96},  { 3.13,27.63},{32.14,75.94},{35.33,81.42},{22.10,65.27},  {91.51,133.81},{93.84,132.45},{ 3.26,23.94},{18.72,41.72},  {86.16,123.78},{73.34,93.57},{50.96,74.51},{ 3.74,20.27},  {30.91,86.97},{25.75,81.03},{80.51,112.91},{99.33,129.01},  { 2.78,37.66},{ 1.13,40.07},{45.63,95.65},{29.63,78.53},  {98.89,150.89},{96.92,157.49},{99.71,151.04},{83.86,120.28},  {23.49,61.26},{10.75,43.94},{97.28,144.71},{64.10,94.47},  {42.54,87.88},{ 4.66,52.51},{38.58,75.97},{77.72,115.77},  {64.94,93.94},{66.24,105.03},{70.86,115.60},{ 4.14,46.01},  {34.88,47.03},{67.73,100.39},{94.56,142.78},{34.32,69.74},  {31.02,70.75},{69.03,107.76},{50.35,96.39},{40.21,71.02},  {23.46,59.37},{38.40,58.18},{69.95,104.47},{97.44,143.49},  {17.96,40.47},{74.16,110.72},{26.44,78.39},{58.17,96.32},  {95.98,137.43},{17.22,46.85},{56.88,85.36},{77.06,134.15},  {58.65,96.58},{89.74,144.56},{23.13,48.01},{ 6.08,53.04},  { 7.92,47.13},{ 6.74,33.20},{31.71,63.90},{ 8.57,42.10},  {39.37,83.68},{80.15,126.22},{99.07,138.57},{30.34,69.90},  {75.37,122.73},{35.56,60.71},{64.67,101.00},{30.87,57.49},  {93.68,145.04},{59.37,103.00},{ 2.66,24.57},{34.73,66.59},  {31.61,77.16},{22.69,50.47},{25.86,46.76},{41.39,79.20},  {26.70,67.54},{62.46,100.54},{71.81,90.95},{50.58,84.93},  {25.58,68.30},{73.31,109.45},{83.89,140.03},{21.26,40.12},  {36.64,71.54},{27.30,60.72},{85.99,131.70},{34.10,82.07},  { 8.17,39.53},{71.88,120.91},{50.87,83.25},{69.88,95.21},  {91.74,121.85},{95.47,138.41},{40.21,87.08},{12.36,55.26},  {14.70,50.16},{60.71,103.15},{80.63,131.65},{ 3.15,30.58},  { 7.34,47.98},{ 8.98,27.18},{71.46,104.36},{ 4.32,37.76},  {39.36,76.23},{ 1.26,23.68},{21.06,61.51},{ 1.50,39.92},  {42.60,81.07},{46.54,82.57},{21.96,55.45},{42.59,63.96},  {61.06,101.39},{14.22,48.51},{56.16,90.04},{39.19,69.37},  {34.86,67.85},{79.53,120.39},{29.67,75.34},{51.66,77.48},  {65.68,105.18},{45.90,73.86},{ 9.47,24.89},{32.49,87.05},  {26.80,80.19},{77.57,122.81},{ 0.27,60.41},{92.54,122.42},  { 3.20,32.32},{12.52,43.48},{94.09,147.67},{38.29,53.35},  {94.96,137.92},{85.01,99.90},{15.46,39.13},{ 8.16,38.50},  {65.88,101.52},{40.33,75.48},{43.60,85.09},{79.55,111.11},  {73.38,96.56},{25.49,55.39},{41.85,85.10},{35.83,81.47},  {17.51,41.38},{52.90,99.47},{46.02,89.72},{41.51,102.94},  { 6.69,48.77},{39.31,72.57},{89.92,154.36},{79.44,115.66},  {93.00,146.67},{93.80,136.18},{44.52,78.02},{95.82,144.52},  {66.98,111.39},{62.00,109.03},{64.17,99.63},{ 6.69,52.44},  {76.69,122.70},{86.97,147.68},{ 5.62,53.68},{37.36,72.20},  {20.42,61.81},{18.51,54.44},{31.34,71.19},{19.11,52.05},  {14.73,40.57},{13.92,67.50},{ 9.57,40.88},{59.43,79.52},  {94.52,155.73},{26.97,59.84},{14.54,43.10},{ 6.72,48.19},  {90.36,146.08},{ 7.03,41.80},{96.29,128.34},{ 5.92,52.98},  {12.61,56.44},{79.35,136.81},{91.68,125.61},{13.76,30.63},  {36.92,81.04},{84.75,154.73},{62.52,118.08},{17.74,54.27},  {40.27,79.18},{13.97,53.91},{36.23,66.83},{27.99,49.55},  {47.59,89.62},{ 3.45,12.53},{14.47,56.37},{76.79,133.19},  {63.41,94.95},{ 2.86,24.22},{32.87,72.92},{70.77,103.55},  {26.83,48.72},{82.62,107.53},{22.39,71.04},{47.50,63.00},  {83.19,120.01},{41.59,78.88},{87.92,132.04},{34.18,81.49},  {33.22,90.14},{57.76,101.46},{77.89,120.45},{49.05,77.48},  {66.98,110.78},{11.17,48.20},{78.44,108.70},{59.79,127.63},  {93.87,120.78},{ 1.45,36.16},{52.96,73.31},{17.40,44.38},  {97.22,135.64},{53.73,81.45},{76.18,123.21},{30.42,73.39},  {36.25,71.59},{ 5.19,31.68},{43.86,71.53},{61.01,101.69},  {92.16,142.72},{71.31,130.03},{79.94,114.88},{16.48,57.55},  { 4.25,31.89},{20.33,55.04},{55.25,102.92},{14.94,44.18},  {87.78,116.28},{87.59,119.95},{27.43,54.72},{20.43,71.30},  { 9.04,43.69},{78.75,104.32},{59.98,100.13},{84.71,123.62},  { 1.30,24.41},{83.81,129.07},{23.39,66.21},{42.82,79.97},  {53.34,104.34},{65.49,119.13},{54.07,94.88},{44.57,74.05},  {80.26,129.86},{38.64,63.87},{41.03,68.38},{94.67,141.70},  {13.69,38.65},{74.79,112.56},{70.51,115.89},{85.51,128.68},  {97.26,140.91},{82.02,151.28},{ 9.02,50.71},{16.63,55.01},  {61.24,80.17},{99.50,142.58},{16.50,38.78},{69.13,117.24},  {12.46,37.01},{41.72,71.49},{23.39,39.35},{31.79,54.32},  {44.61,88.22},{18.65,49.73},{19.65,41.74},{15.05,45.89},  {40.56,71.03},{63.95,116.95},{ 8.62,39.32},{48.86,88.76},  {58.54,93.44},{13.85,47.33},{60.42,89.67},{84.69,140.35},  {14.08,51.78},{37.83,78.89},{84.58,130.84},{51.02,71.48},  { 8.79,43.19},{95.58,149.46},{74.16,115.02},{55.45,105.22},  {95.01,128.69},{61.62,98.46},{54.07,80.18},{84.40,128.05},  {98.40,137.74},{61.00,101.13},{20.63,54.26},{46.63,96.72},  {15.89,52.46},{ 8.26,35.59},{54.78,87.95},{89.81,133.64},  {39.15,66.33},{94.49,127.07},{85.78,127.93},{94.91,145.32},  { 8.70,34.57},{91.00,124.91},{22.42,66.83},{97.97,162.26},  {96.58,134.70},{72.66,112.00},{92.71,137.11},{72.43,113.30},  {58.55,94.26},{53.13,76.54},{55.50,100.15},{33.32,79.67},  {31.47,55.62},{12.99,57.24},{18.29,41.81},{60.32,103.09},  {51.37,83.11},{68.75,101.36},{47.84,54.77},{25.72,68.03},  {69.70,114.51},{20.36,55.10},{78.08,125.78},{35.27,70.19},  {93.13,139.28},{29.60,50.01},{43.45,72.80},{56.11,98.12},  {62.69,105.03},{71.69,112.93},{34.79,48.57},{69.79,105.35},  { 9.44,26.26},{90.57,122.21},{86.12,122.68},{64.32,115.35},  {80.37,117.95},{60.31,90.09},{47.44,83.79},{10.34,38.57},  {96.74,133.19},{94.56,157.85},{75.67,114.07},{17.31,51.88},  { 6.37,33.90},{33.31,73.68},{22.93,76.97},{87.68,136.16},  {78.65,119.82},{74.05,111.98},{79.37,110.40},{56.95,100.33},  {97.45,143.16},{30.11,83.98},{29.17,58.00},{98.08,133.71},  { 1.65,30.80},{25.49,60.59},{32.13,44.10},{21.33,72.68},  {28.16,51.30},{59.04,87.67},{ 0.79,42.10},{31.03,68.59},  {72.16,125.28},{93.83,134.80},{25.03,63.16},{29.69,64.31},  {13.77,59.38},{45.95,91.38},{81.22,109.99},{48.54,88.83},  {62.84,87.41},{ 0.36,24.14},{ 4.22,23.71},{19.12,68.78},  {95.07,132.09},{18.72,57.73},{59.25,89.61},{39.64,61.27},  {41.61,61.23},{96.30,134.31},{47.02,90.94},{77.25,97.69},  {71.41,94.11},{33.74,93.27},{28.72,66.91},{35.69,69.99},  {19.79,52.89},{53.62,72.52},{44.74,82.39},{45.06,98.87},  {66.65,119.64},{68.70,105.26},{42.56,65.63},{56.87,97.73},  {31.58,79.47},{94.02,136.37},{13.60,33.19},{71.89,105.52},  {22.90,42.16},{28.51,64.91},{64.97,111.95},{61.23,88.54},  {41.57,77.14},{34.50,50.93},{90.50,131.48},{91.31,136.80},  {41.88,68.18},{90.25,119.85},{55.16,99.86},{14.62,22.68},  {65.97,116.03},{10.45,66.45},{76.43,109.15},{78.49,121.95},  { 5.08,39.65},{82.21,143.61},{47.78,70.97},{93.29,120.18},  {32.55,73.97},{31.08,67.44},{94.39,139.58},{64.99,106.12},  {68.52,114.77},{39.32,74.75},{64.08,93.46},{96.53,161.32},  {98.39,156.92},{71.18,107.82},{41.98,69.10},{35.65,82.51},  {34.74,53.68},{69.85,108.65},{93.04,145.91},{75.90,125.93},  {65.63,104.27},{66.59,109.75},{ 6.98,40.88},{ 1.40,28.66},  { 7.18,23.80},{ 0.46,31.77},{51.14,85.15},{16.70,46.46},  {87.30,124.42},{30.33,80.85},{97.49,140.48},{16.52,37.74},  {78.86,103.99},{87.10,120.00},{29.19,73.87},{42.17,88.73},  { 6.46,47.30},{15.45,43.73},{28.68,70.84},{42.53,94.88},  {27.49,73.47},{75.78,135.66},{ 2.17,23.87},{86.14,125.27},  {48.72,87.89},{60.13,97.58},{98.30,145.84},{58.31,105.87},  {84.27,131.24},{49.79,94.46},{89.31,136.14},{83.91,124.03},  {18.93,44.20},{59.13,87.87},{49.61,85.82},{53.11,101.41},  {93.86,131.36},{34.04,68.21},{ 5.92,41.12},{22.82,75.35},  {32.25,87.50},{29.39,59.33},{70.83,127.40},{15.00,51.39},  {69.16,126.06},{51.04,76.62},{33.96,79.15},{98.87,140.31},  { 2.69,30.25},{37.36,88.30},{22.90,66.20},{60.87,107.20},  {10.58,35.55},{74.31,107.46},{78.11,106.57},{85.38,129.68},  {86.06,124.62},{14.02,38.35},{32.12,55.18},{66.83,102.39},  {41.95,86.13},{43.32,78.87},{ 1.92,28.79},{29.17,46.33},  {52.56,98.99},{68.47,113.85},{45.06,95.08},{33.45,93.28},  {55.80,96.01},{24.67,60.35},{26.33,65.37},{67.49,108.00},  {41.86,69.13},{71.83,116.90},{ 9.58,51.97},{66.94,120.08},  {42.27,84.89},{74.15,128.39},{70.21,128.74},{ 8.49,48.64},  {45.98,82.55},{21.73,53.24},{27.99,68.32},{65.48,109.73},  {29.77,74.81},{80.12,130.03},{81.44,112.31},{96.72,149.06},  {33.81,83.49},{ 4.14,17.66},{ 9.77,57.10},{23.96,42.28},  {47.06,88.64},{44.96,81.04},{ 7.72,41.74},{71.48,117.15},  {64.34,102.61},{35.46,89.47},{39.74,77.99},{64.32,107.97},  {75.03,123.09},{35.43,79.58},{70.86,127.89},{15.64,28.64},  {85.00,102.99},{ 9.67,45.55},{10.70,43.05},{ 9.13,31.71},  {49.50,87.28},{29.10,64.51},{73.76,125.71},{13.75,54.78},  {95.20,138.81},{77.96,112.16},{76.43,111.70},{14.84,56.07},  {44.13,92.19},{25.03,73.47},{ 2.69,39.65},{70.25,100.08},  {41.50,75.66},{68.17,122.55},{49.02,102.83},{76.44,115.87},  {16.37,66.78},{85.55,124.31},{19.85,34.71},{89.74,140.16},  {24.33,41.24},{37.07,86.95},{92.51,133.66},{91.12,119.68},  {23.27,62.87},{61.76,93.53},{11.79,50.05},{87.44,113.21},  {91.22,119.00},{55.71,83.98},{34.90,83.25},{ 9.35,33.05},  {66.54,99.59},{14.00,58.87},{80.35,123.07},{29.06,37.69},  {21.60,72.09},{ 7.08,30.13},{57.28,97.62},{64.19,94.55},  {52.38,93.80},{89.49,160.44},{38.16,63.02},{51.21,88.27},  {95.14,132.45},{16.85,40.33},{31.98,56.20},{90.29,127.04},  {96.35,156.03},{34.37,82.14},{84.28,117.89},{10.33,64.50},  {24.54,64.56},{69.70,100.33},{72.97,104.79},{97.49,150.73},  {95.27,144.84},{79.25,111.45},{18.04,54.13},{ 4.49,28.53}  }; |

Running the posix Version 10 times.

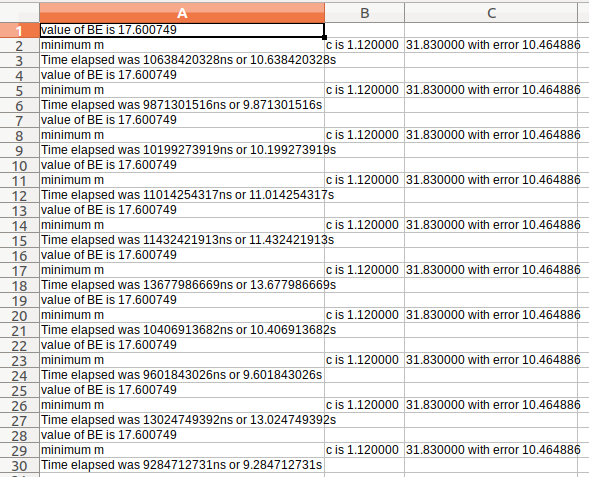


Figure Compiling and running the posix version 10 times

Figure 6Output of ten runs

Here is table for ten times run of the posix program along with the M and C.   
Mean running time for the program as calculated is:

MRTlrp= **∑ xi/ n**

= 3.2716375805 second is the mean run time   
since mean runtime needs to be calculated as rate the harmonic mean run time would be:

MRTlrp Harmonic= n / ∑ [1/Xi]

hence,

del(execution time)= 3.271607871 second per program.

**Comparisons table**

|  |  |
| --- | --- |
| **Sequential RUN** | **Posix RUN** |
| **Mean Execution time (Average Execution)=** 0.119651439s | **Mean Execution Time (Average Execution)=** 3.2716375805s |
| **Rate Of execution per program=** 0.119607871s | **Rate of Execution Per program=** 3.271607871s |

Here analysis of the Mean Execution time and rate of execution per program shows that the time taken for the execution of sequential version was less than that compare to Posix run. This shows a similar behaviour of batch job execution by as analysed and explained on the Image Processing hence for optimization another replacement technique is suggested.

# CUDA

CUDA (Compute Unified Device Architecture) is a optimization platform released and maintained by the GPGPU manufacturers NVidia. It is also the most efficient way for thread processing model till date. The real **Pseudo Parallelism** can be demonstrated using CUDA and NVidia GPGPU which provides a smooth device and host communication. Cuda works with a general transistor Single Instruction Multiple Data architecture which means multiple cores in the GPU can execute multiple state single batch jobs all at once which do not require any scheduling.

Some general use CUDA functions are \_\_host\_\_identifier , \_\_device\_\_identifier, cudaMalloc() cudaMemcpy() etc….

Cuda have its own technique for data serialization for processing it into threads. It uses dimensional blocks for processing the inputs into threads. These blocks are managed by the cuda processor using dimension indexing techniques (indexed as 1D,2D,3D on a general vector space).

## Password Cracking

Paste your source code for your CUDA based password cracker here

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

Write a short analysis of the results

## Image Processing

For image processing with CUDA the algorithm was modified with the cuda functions cudaMemcpy() , cudaMalloc() to copy and allocate the processing data and to fetch those data for respective cuda indexed threads. Here as an input we use the same 2D image matrix used for user level parallelism (pOSIX) and Sequential Execution algorithm. The input matrix having 7200 pixels is processed unto 7200 CUDA threads. The input is then copied from host to device using the kernel function while the kernel function executes the each pixel referencing to their allocated indexed dimensions.

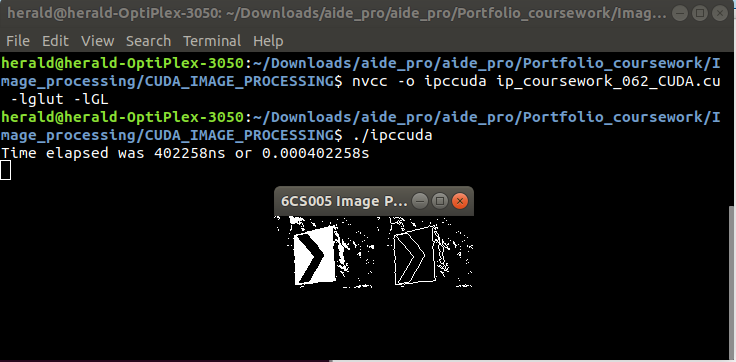
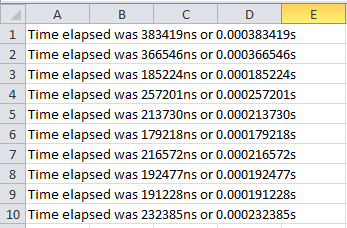


Figure Cuda Simple Run

Below in the source code below the demonstration of cudaMemalloc can be done which is used to dynamically allocate device variables. Similarly the block dimension and thread dimension to process the height and width of the image is donne using block\_Dim() and thread\_Dim(). cudaMemcpy is used to copy the input from host to device and CudaThreadSyncronize() is used to synchronize all 7200 cuda threads. At last to free the var memory on the GPU cudaFree is used for Sterilization.

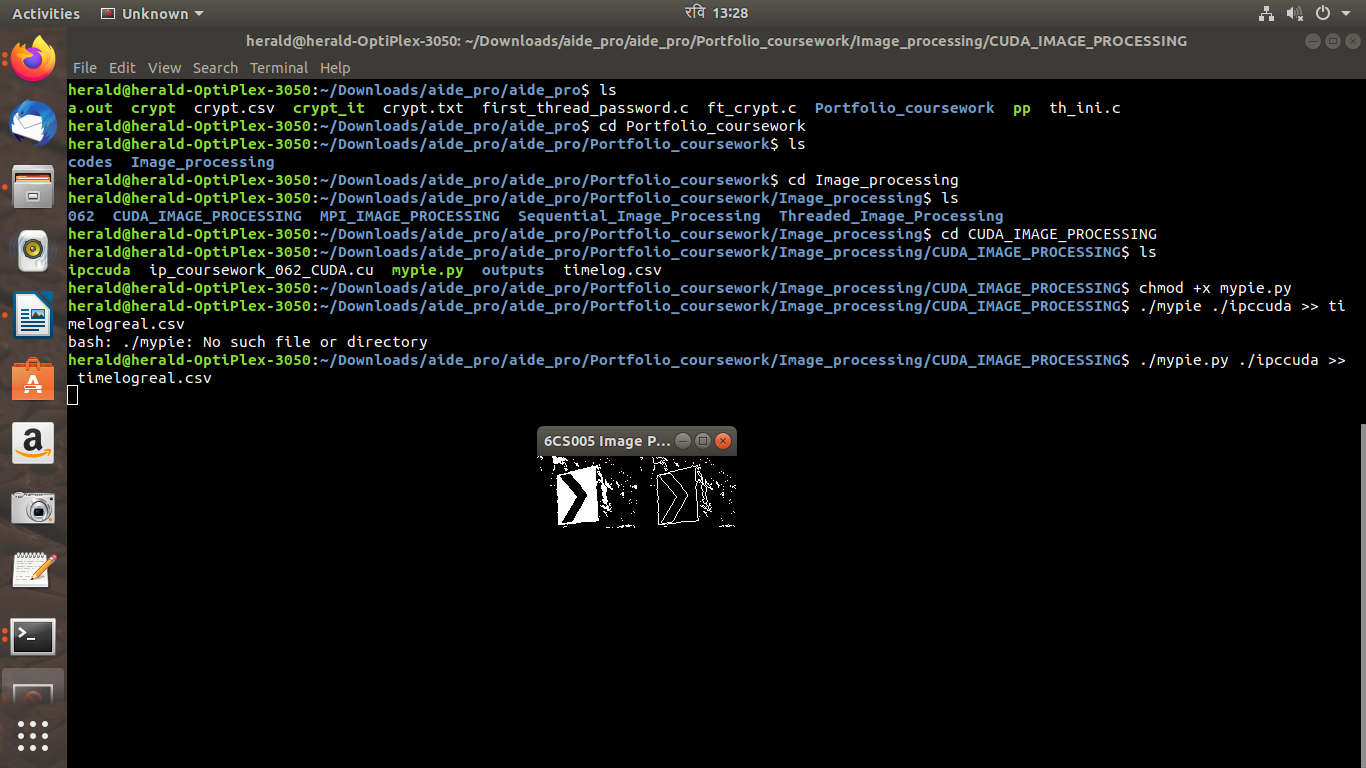
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <GL/glut.h>  #include <GL/gl.h>  #include <malloc.h>  #include <signal.h>  #include <cuda\_runtime\_api.h>  #define WIDTH 100  #define HEIGHT 72  #define PIXEL\_SIZE WIDTH\*HEIGHT  unsigned char results[PIXEL\_SIZE];  //array matrix of imgae provided  unsigned char image[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,  255,0,0,0,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,0,0,  255,255,255,255,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,0,255,255,255,255,255,255,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,  255,255,255,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,255,255,0,0,0,255,0,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,  255,0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,0,0,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,255,255,255,255,255,0,0,0,0,255,  255,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,  0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,  255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,  0,0,255,0,0,0,0,255,0,0,0,0,0,0,255,0,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,255,255,255,255,0,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,0,255,  255,255,255,0,255,255,0,0,0,0,0,255,255,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,255,255,0,255,255,255,255,0,0,255,0,0,0,0,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,0,255,255,0,0,255,255,0,0,0,0,0,0,0,0,  0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,255,255,0,0,0,  0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,0,0,  255,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  0,255,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,255,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,  0,0,0,255,255,0,0,0,0,0,0,0,0,0,0,255,255,255,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,255,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  0,0,255,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,255,0,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  0,0,255,255,255,255,0,255,0,0,0,0,0,0,0,0,0,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,255,255,0,255,255,255,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,255,255,255,0,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,0,0,0,0,  0,0,0,0,0,255,255,255,255,255,0,255,255,255,255,0,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,255,255,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  };  int time\_difference(struct timespec \*start, struct timespec \*finish,  long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  \_\_global\_\_ void detect\_edges(unsigned char \*input\_image\_pixel, unsigned char \*output\_image\_pixel) {  //calculating address for pixel to be processed  int i = (blockIdx.x \* blockDim.x) + threadIdx.x;  int x, y;  int b, d, f, h;  int r;    y = i / WIDTH;  x = i - (WIDTH \* y);  if (x == 0 || y == 0 || x == WIDTH - 1 || y == HEIGHT - 1) {  output\_image\_pixel[i] = 0;  } else {  b = i + WIDTH;  d = i - 1;  f = i + 1;  h = i - WIDTH;    //applyinf filter matrix for filtering  r = (input\_image\_pixel[i] \* 4) + (input\_image\_pixel[b] \* -1) + (input\_image\_pixel[d] \* -1) + (input\_image\_pixel[f] \* -1)  + (input\_image\_pixel[h] \* -1);  if (r > 0) { // if the result is positive this is an edge pixel  output\_image\_pixel[i] = 255;  } else {  output\_image\_pixel[i] = 0;  }  }  }  void tidy\_and\_exit() {  exit(0);  }  void sigint\_callback(int signal\_number){  printf("\nInterrupt from keyboard\n");  tidy\_and\_exit();  }  static void display() {  glClear(GL\_COLOR\_BUFFER\_BIT);  glRasterPos4i(-1, -1, 0, 1);  glDrawPixels(WIDTH, HEIGHT, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);  glRasterPos4i(0, -1, 0, 1);  glDrawPixels(WIDTH, HEIGHT, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);  glFlush();  }  static void key\_pressed(unsigned char key, int x, int y) {  switch(key){  case 27:  tidy\_and\_exit();  break;  default:  printf("\nPress escape to exit\n");  break;  }  }  int main(int argc, char \*\*argv) {    //defining block and thread dimension  dim3 block\_Dim(WIDTH,1,1),thread\_Dim(HEIGHT,1,1);    unsigned char \*d\_results;  unsigned char \*d\_image;      //device memory allocation for device variables using error checking  if(cudaMalloc((void\*\*)&d\_image, sizeof(unsigned char) \* PIXEL\_SIZE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    if(cudaMalloc((void\*\*)&d\_results, sizeof(unsigned char) \* PIXEL\_SIZE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    //copying input image pixel data from host to device  if(cudaMemcpy(d\_image, &image, sizeof(unsigned char) \* PIXEL\_SIZE, cudaMemcpyHostToDevice))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    signal(SIGINT, sigint\_callback);    struct timespec start, finish;  long long int time\_elapsed;  clock\_gettime(CLOCK\_MONOTONIC, &start);  detect\_edges<<<block\_Dim,thread\_Dim>>>(d\_image, d\_results);    //barier point for threads  cudaThreadSynchronize();  //copying output image pixel data from device to host  cudaMemcpy(&results, d\_results, sizeof(unsigned char) \* PIXEL\_SIZE, cudaMemcpyDeviceToHost);    clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &time\_elapsed);  printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,  (time\_elapsed/1.0e9));    //freeing of allocated device memory  cudaFree(d\_image);  cudaFree(d\_results);  glutInit(&argc, argv);  glutInitWindowSize(WIDTH \* 2,HEIGHT);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);    glutCreateWindow("6CS005 Image Progessing Courework using CUDA");  glutDisplayFunc(display);  glutKeyboardFunc(key\_pressed);  glClearColor(0.0, 1.0, 0.0, 1.0);  glutMainLoop();  tidy\_and\_exit();    return 0;  } |

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



Above table shows the run time for execution of Linear Regression algorithm on CUDA GPGPU.

The algorithm was run ten times using a python script.  
**Analysis of runtimes for CUDA and original program.**



Mean running time for the CUDA program as calculated is:

MRTlrcu= **∑ xi/ n**

= 241800 nanosecond is the mean run time = 0.0002418 seconds  
since mean runtime needs to be calculated as rate the harmonic mean run time would be:

MRTlrcu Harmonic= n / ∑ [1/Xi]

= 226105.457872 nano seconds = 0.00022610545seconds

hence,

del(execution time)= 0.00001569455 second per program.

**Comparisons table**

|  |  |
| --- | --- |
| **Sequential RUN** | **CUDA RUN** |
| **Mean Execution time (Average Execution)=** 0.000138324s | **Mean Execution Time (Average Execution)=** 0.0002418s |
| **Rate Of execution per program=** 0.000136542s | **Rate of Execution Per program=** 0.00022610545seconds |

Hence del (time) wrt CUDA execution can be calculated as

= Mean Execution Time for CUDA - Mean Execution time for sequential = 0.000103476s

Since we know,

(Mean Execution time for CUDA) <<<< (Mean Execution Time for sequential processing)

The demonstration of ***pseudo parallelism*** not only proved to be less time consuming it also proved that it is much better option in compare to using user level parallelism ***i.e POSIX threads***.

## Linear Regression

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

Write a short analysis of the results

# MPI

## Password Cracking

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

Paste your source code for your MPI based password cracker here

Write a short analysis of the results

## Image Processing

Paste your source code for your MPI based image processor

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

Write a short analysis of the results

## Linear Regression

Paste your source code for your MPI based linear regression

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

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

Write a short analysis of the results

# Verbose Repository Log

Paste your verbose format repository log here. With subversion this can be achieved by the following:

svn update

svn –v log > log.txt

gedit log.txt

Then select, copy and paste the text here