6CS005 Learning Journal - Semester 2 2018/19

Put your name and student number here

# Table of Contents

[Table of Contents 1](#_Toc7965571)

[1 POSIX Threads 4](#_Toc7965572)

[1.1 Password Cracking 5](#_Toc7965573)

[1.2 Image Processing 19](#_Toc7965574)

[1.3 Linear Regression 39](#_Toc7965575)

[2 CUDA 61](#_Toc7965576)

[2.1 Password Cracking 61](#_Toc7965577)

[2.2 Image Processing 61](#_Toc7965578)

[2.3 Linear Regression 62](#_Toc7965579)

[3 MPI 62](#_Toc7965580)

[3.1 Password Cracking 63](#_Toc7965581)

[3.2 Image Processing 63](#_Toc7965582)

[3.3 Linear Regression 64](#_Toc7965583)

[4 Verbose Repository Log 64](#_Toc7965584)

This document contains evaluation report on implementation of different **parallel computing** paradigms such as **POSIX**, **MPI** and **CUDA** as well as **serial computing** in three different problem domains: **Password Cracking**, **Image Processing** and **Linear Regression**. The PC used as a **node** is equipped with **shared memory architecture**. A particular jargon used for this module is **high performance computing (HPC)**. High performance computing is all about solving large and complex problems using parallel computing technology in least possible time with high accuracy.

**Serial Computing**

In serial computing, a computational problem is solved using software that contains series of instructions which gets executed sequentially i.e. one after another. Those software run on a single processor and more importantly only one instructions from the software is executed at a time.

(Flynn, 1966) serial computing with multi processor computers can have **SISD (Single Instruction stream Single Data stream).** This means only one particular instruction from software gets executed and only one particular data is operated on by the instruction at a time.

**Parallel Computing**

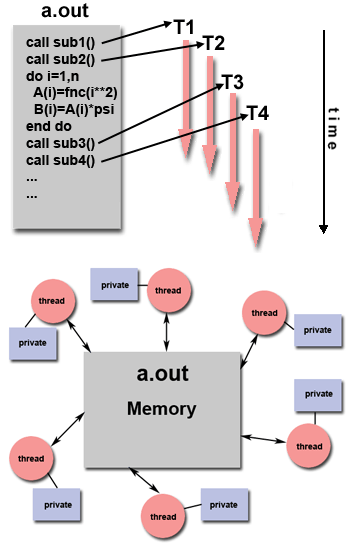
Parallel computing is solving computational problem concurrently by dividing the problem among multiple computing resources (e.g. CPU, system memory, processes etc). Computers equipped with multi-core processors or multiple processors and shared memory architecture are capable of performing parallel computing.

**Shared Memory Architecture**

In this architecture, all the CPUs connected to the system are capable of accessing same memory as global address space. More precisely, those processors can operate independently but share the same memory resources. Common features of this architecture are as follows:

* Changes in data in memory location made by one process/processor are visible to other process/processors and those processors can read most recently updated data.
* Since memory is shared; sharing of data between processes becomes fast and uniform.

**Different parallel programming models**

There are different parallel programming models and commonly used are as follows:

1. **Threads Model**

Thread model works can on PCs with shared memory architecture. According to this model, a program is initiated by a single “heavy weight” process. The execution of the program then branches out to different sub-execution path termed as “threads”. Threads are considered light out since they are not resource intensive as the main process of the program. The threads created have their own local data but also shares the resources allocated for the main process. This avoids overhead concerned with replicating a program’s resources by each threads.

Threads created are scheduled and run by the host Operating System. Nevertheless, they are capable of executing concurrently and independently. Threads created perform their job and exit but the main process may remain for longer period of time in memory.

Two different thread technology used in this module are:

1. POSIX thread programming
2. CUDA GPU thread programming

Figure : basic thread model (Barney, 2018)

# POSIX Threads

**POSIX (Portable Operating System Interface for UNIX based System) Thread**

Initially, the thread models used to be vendor specific or proprietary. Different hardware vendors have their own implementation of thread models. Those different thread models served as a prominent difficulty for the programmers to develop portable thread based applications. To overcome such situation, IEEE developed a standard for the thread model in 1995 which is known as **IEEE POSIX 1003.1c** standard**.**

POSIX library is based on C language. This library allows programmers to write software as multithreaded application. Since C is a system programming language, multithreaded application written using POSIX is efficient. A thread in multithreaded program written using POSIX library is called **pthread**.

In shared memory architecture multithreaded program is implemented by the processors using CPU time slicing in processor cores.

|  |  |
| --- | --- |
|  | |
|  |  |

## Password Cracking

**Serial version Password Cracking – Meantime calculation**

Sequential version of the password cracking program was run 10 times for the set of 4 different passwords each consisting of 2 uppercase Alphabets initials and 2 digit numbers. Following is the table showing execution time log and mean time in seconds for the run:

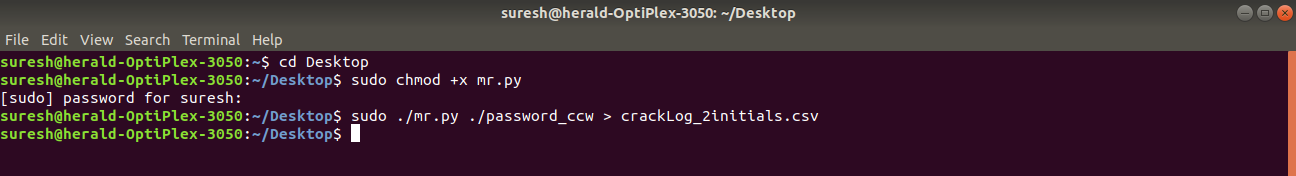
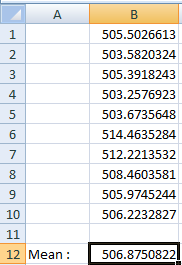


Figure running password cracking program 10 times

The output of the program was directed to **crackLog\_2initials.csv** file. Then only time taken by each iteration was exported to ‘**log\_pccw\_seq.csv**’ file. The meantime was then calculated using average() in Libreoffice Calc.



The mean execution time for cracking the set of 4 passwords came to be **506.87 seconds** or **8.4 minutes** which also means it took **2.1 minutes** for cracking a password.

**Note:** For better readability of the output and locate the results, the printf statement for printing unmatched combinations of the password in the program has been commented out. And this definitely helps program to do its tasks at reduced amount of time by avoiding potential **bottleneck**.

**Estimation of execution time – 3 initials and 2 digit password (serial version)**

Each of the password consisted of 2 uppercase alphabets as initials and 2 digit numbers and there were 4 such passwords. The sequential version of the program made combinations of initials and digits using nested FOR loops.

Mathematically, according to permutation and combination rules, there would be **26\*26\*100=67600** different combinations that the program has to check. Those combinations are generated and checked by 3 nested FOR loops this means that the loop iterates total 67600 times while cracking a passwords. The program cracks 4 such passwords then

Mathematically,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of combinations** | **Number of nested loops** | **Complexity order** | **Number of loop iterations** | **Number of passwords to crack** | **Total time taken** |
| 26\*26\*100=67600 | 3 | O(n3) | 67600 times | 4 | 8.4 minutes (on average) |
| 26\*26\*100=67600 | 3 | O(n3) | 67600 times | 1 | (8.4/4)=2.1 minutes (on average) |

Upon adding extra alphabet initial to the password, extra outermost loop has to be added for making combinations then

Mathematically, using unitary methods,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of combinations** | **Number of nested loops** | **Complexity order** | **Number of loop iterations** | **Number of passwords to crack** | **Probable Required time** |
| 26\*26\*100=67600\*26 | 4 | O(n4) | 1757600times | 4 | 8.4 \*26 = 218.4 minutes (on average) |
| 26\*26\*100=67600\*26 | 4 | O(n4) | 1757600times | 1 | (8.4/4)=2.1\*26=54.6 minutes (on average) |

Thus assuming that rest of the statements such as variable declaration and initialization taking constant time for execution in both cases, it is estimated to take **218.4** **minutes or 3.6 hours** for the same computer to decrypt 4 passwords consisting of **3 alphabet initials and a two digit number.**

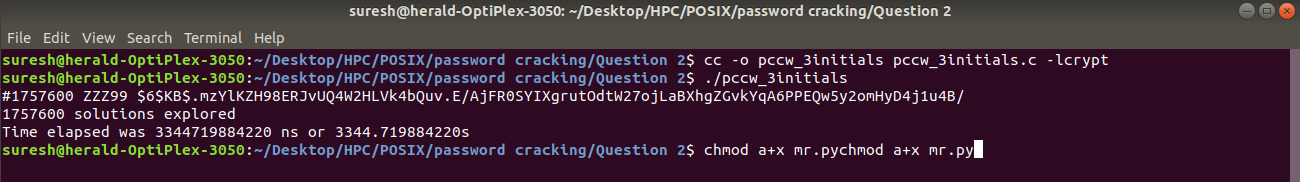
This concludes that greater the number of characters in password greater is the time required for cracking the passwords for computing machine.

**Code for cracking password – 3 initials and 2 digits**

|  |
| --- |
| #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;  } |

**Estimated execution time VS Actual execution time – A 3 initials 2 digit password cracking program**

The modified program was run on the same computer and following output was observed.



The program took **3344.7 seconds** or **55.7 minutes** to crack a single 3initials 2 digit password but the estimated time for the program was **54.6 minutes**.

Apparently,

**Actual execution time >** **Estimated execution time**

This is obvious because the PC is using shared memory architecture. Multiple cores are using the same memory and many other processes or programs are being run at the same time. The processor needs to schedule the processes it runs and it uses time slicing and process swapping form core to core inside the system. All these steps requires some amount of time. This time has added to our estimated time and hence actual time surpassed the estimated time for the program.

**Multithread Version using POSIX – Password Cracking**

**Data partitioning –domain decomposition into blocks**

Data partitioning or domain decomposition is dividing problem into discrete chunks which can be assigned to the threads or processes. Before writing code it is necessary to divide the problem for two threads equally achieving **load balancing** among those threads.

Since password cracking program checks for 67600 combinations, first thing to do is to divide the number of combinations to be checked into two equal halves and assign them to the different threads. On partitioning domain data into blocks we observe,

|  |  |  |
| --- | --- | --- |
| **Thread** | **Block** | **Number of combination to be checked** |
| 1 | AA00 to MM99 | 33800 |
| 2 | NN00 to ZZ99 | 33800 |

**Synchronization – using Mutex Lock**

Mutex stands for mutual exclusion. Mutex ensures safe access to the shared resource. For this purpose, some operations performed by threads or processes are critical and must be serialized i.e. although threads run in parallel, when it performs a section of code, one needs to have monopoly while executing that section of code, this effect is achieved by one of the synchronization techniques in POSIX – Mutex Lock. The first thread reaching the section of the critical code sets the lock and performs the specified operation and releases its lock as soon as it completes its task. The next thread then does the same thing. This trend continues so forth and so on.

|  |
| --- |
| #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$PUtGvfAAHdh8/92SCkEAjarFqQI6k3e.3e9t8lZ1MN/wY5qvIG.RchUF4VzV6uKJDZL1up/s95NWn82Fz/FQR.",  "$6$KB$u98pcr1UEAEX25XrSrB2Eakm846e83r/lrp2REKCG5o/rphTou9l.3GiEFzchEHZAN9hoAYBlT/sMhr4RxFeL/",  "$6$KB$7JS8Bt9NYczSqQEpWl/8IMZDD4yIs2kYqUYud.83hoyiSpS6ZFF8SeIFm1P5KZvkA1ytLT8WeNG4l/.yaHiVf1",  "$6$KB$uZeo3hLi9vsKvvY.xhFwqRrLyBCB/v/q9vc8QGCaaCnhU8AkqigCnXBc/cNTM.bBPwSV2.6pNBwAf9p4jLsMC1"  };  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;  } |

**Mean execution time comparison – sequential version VS multithreaded version**

Both of the password carking programs were run 10 times and the following outputs were obtained.

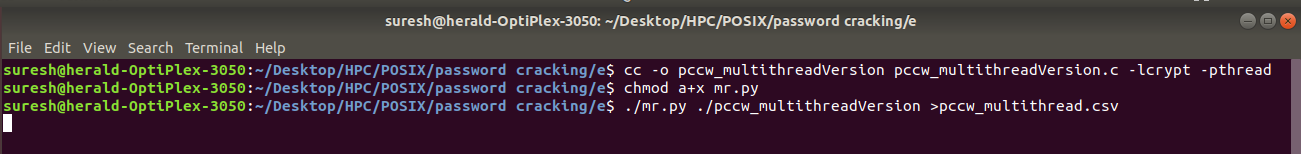


Figure Running of multithread version of password cracking program

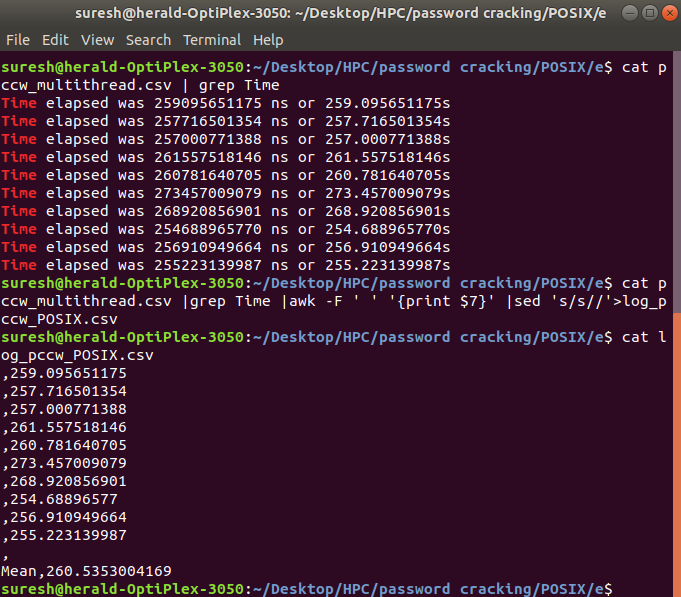


Figure : Mean execution time for multithreaded version of password crack

Multithreaded version of password cracking program took **260.53 seconds or 4.34 minutes** to crack the set of password supplied while sequential version took **8.4 minutes** to resolve the same set of passwords.

Mathematically,

**Mean execution Time (Multithread Version: 4.3 minutes) ≈ Mean execution Time (Sequential version: 8.4 minutes) / 2**

The sequential version took twice the time taken by multithreaded version (with two threads enabled) program. This was because in sequential version a single thread as main ran the program and the program contained a loop which would run **26\*26\*100=67600 times**. Whereas, in case of multithreaded version, each of the two threads would run **13\*26\*100=33800** times i.e. **33800+33800=67600 times** in total but since two threads would run in parallel, it is obvious that the task would be halved by those threads. Hence, mathematically, it would be expected that the multithreaded version of the program (with two threads) to do the password cracking job in **approximately half time** taken by the sequential version of the same program. In conclusion, the reality met with the expectation.

## Image Processing

Following is the image having size 72 \*100 pixels supplied for the edge detection program.



Figure : Input image

**Sequential version – Image Processing program**

The program sequentially processes each of 7200 pixel values defined as dataset in the program. The program feeds each of the pixels to edge detection algorithm and finally displays image with detected edges.

**Output**

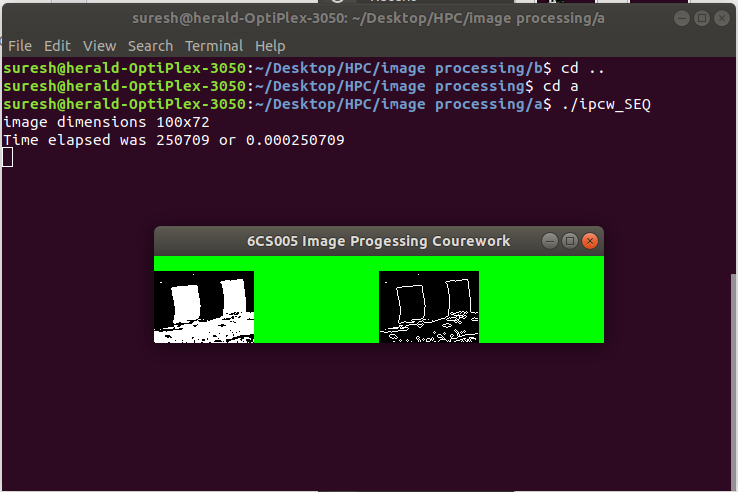
****

Figure : Output sequential version Image processing

**Multithreaded version using POSIX – Image processing program**

**Data partitioning –domain decomposition into Cyclic dataset format**

The dataset is 1D array consisting of 7200 elements and the program needs to have 4 different POSIX threads to process those elementary pixel data. **Load Balancing** is achieved by using striding technique. The logic is similar to dividing entire array into sub array of size 4 and then the thread 1 will always process the 1st indexed element, thread 2 will process 2nd indexed element and so on. This logic achieves load balancing as well because each threads get to process 1800 pixel data in total.

|  |
| --- |
| #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 width 100  #define height 72  #define n 4  #define total\_pixel width\*height  //struct definition for using as argument to the thread function defined  typedef struct {  int start\_index;  int stride;  } thread\_Argument;  //inbuilt structure for reading in second and nanosecond  struct timespec start, finish;  // variable to hold time difference  long long int time\_elapsed;  unsigned char image[], results[width \* height];  //since 4 threads will be used, 4 different thread\_Argument variables are required  thread\_Argument 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){    thread\_Argument \*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 / width;  x = i - (width \* y);  if (x == 0 || y == 0 || x == width - 1 || y == height - 1) {  results[i] = 0;  } else {  b = i + width;  d = i - 1;  f = i + 1;  h = i - width;  r = (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\_elapsed);  printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed,  (time\_elapsed/1.0e9));  }  void tidy\_and\_exit() {  exit(0);  }  void sigint\_callback(int signal\_number){  printf("\nInterrupt from keyboard\n");  tidy\_and\_exit();  }  static void display() {  glClear(GL\_COLOR\_BUFFER\_BIT);  glRasterPos4i(-1, -1, 0, 1);  glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, image);  glRasterPos4i(0, -1, 0, 1);  glDrawPixels(width, height, GL\_LUMINANCE, GL\_UNSIGNED\_BYTE, results);  glFlush();  }  static void key\_pressed(unsigned char key, int x, int y) {  switch(key){  case 27: // escape  tidy\_and\_exit();  break;  default:  printf("\nPress escape to exit\n");  break;  }  }  int main(int argc, char \*\*argv) {  signal(SIGINT, sigint\_callback);    printf("image dimensions %dx%d\n", width, height);  detect\_edges();  glutInit(&argc, argv);  glutInitWindowSize(width \* 2,height);  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;  }  unsigned char image[] = {0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,255,255,255,255,  255,255,255,0,255,255,255,255,255,255,0,0,255,255,255,255,255,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,0,  255,0,0,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,  255,255,0,0,0,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,0,0,0,255,255,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,  255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,  255,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,  255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,0,0,255,255,255,255,255,  255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,255,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,255,255,255,0,0,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,0,255,0,0,0,0,0,255,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  0,0,255,0,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,0,0,0,0,255,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,255,255,255,0,0,0,0,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  }; |

**Output**

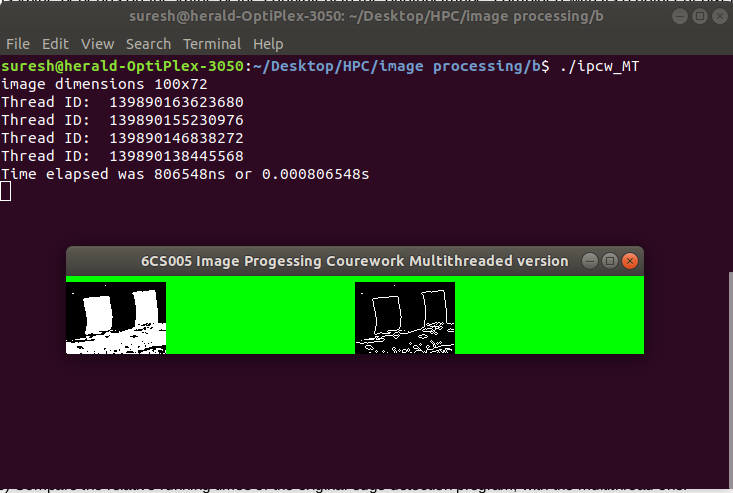


Figure : Output edge detection program multithread version

**Mean execution time – Sequential VS multithreaded version Image processing**

Both sequential and multithread versions of image processing program were run 10 times and mean execution time was calculated as shown below:

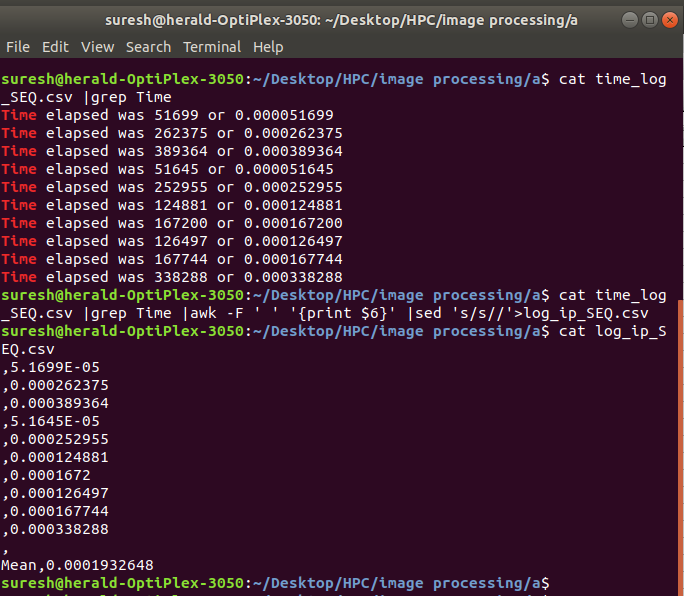


Figure : Mean execution time for sequential version Image processing

Mean execution time for sequential version came to be **0.0001932 seconds** or **.19 milliseconds.**

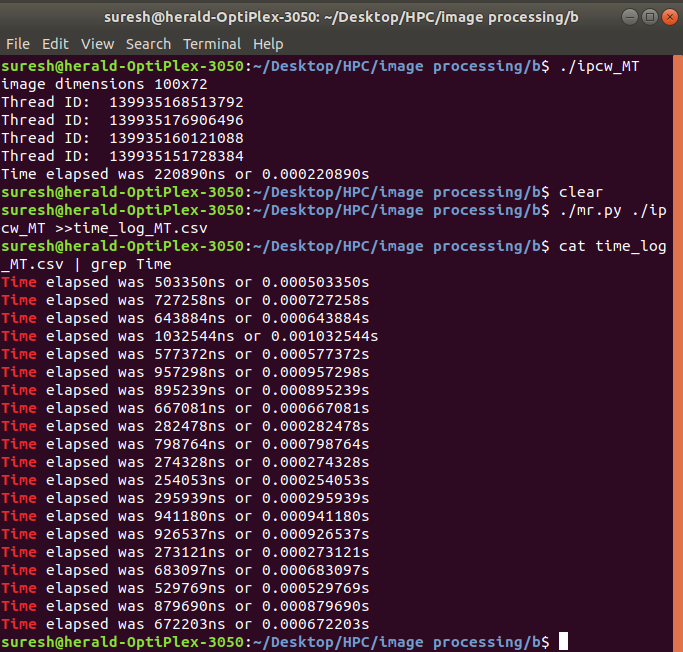


Figure : multithread version image processing program 20 times run

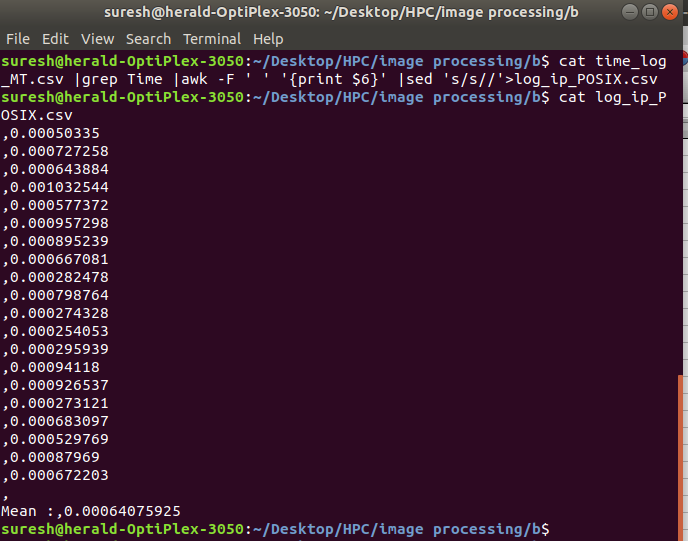


Figure : Mean execution time image processing Multithread version

The mean execution time for multithread version image processing program came to be **0.0006407 seconds or 0.64 milliseconds.**

**Mean Execution time analysis – sequential Vs Multithread image processing**

Mathematically for image processing program,

**Mean execution time Multithread version (0.64 ms) > mean execution time Sequential version (0.19 ms)**

Although it was supposed for multithread version to have less mean execution time reality was opposite. It is because array of size 7200 pixel values and computation of edge did not seem heavy for the processor therefore even normal or sequential flow of the program was able to perform its task in much less amount of time i.e. 0**.19 milliseconds.** The time difference would be prominent if data size would have been of range 1 million or so but the CPU clock frequency is enough faster to process those 7200 pixels and parallelizing such short running tasks can actually result decreased performance because parallelizing requires setting parallel environment e.g. thread creation ,thread scheduling and joining of threads at the end. These tasks requires extra CPU clock cycles to execute and this adds to overall mean execution time and final result is that the time taken by multithread program outweighs the time taken by sequential version of the program.

## Linear Regression

**Scatter Plot**

Dataset on program file lr00.c were replaced with dataset that were included on linear regression program. Then lr00.c was executed and output was directed to a CSV file named ‘lr00\_results.csv’. A jupyter notebook was created and the data-points from the CSV file was read using Pandas and scatter plot was done using Pyplot package from Matplotlib library.

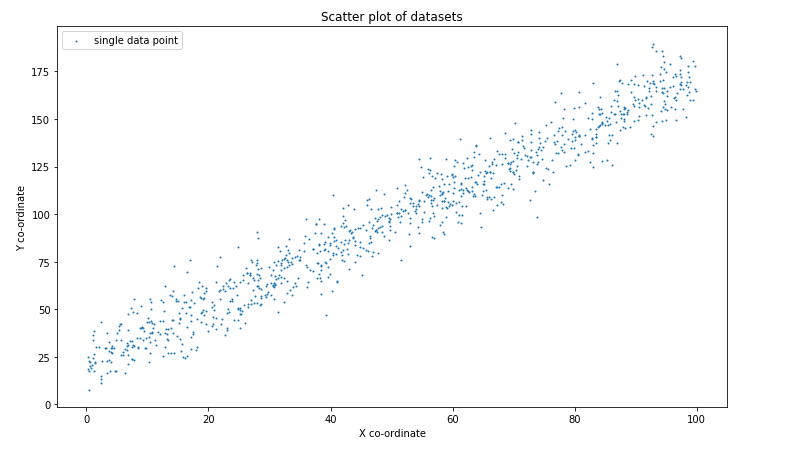


Figure Scatter plot for the given dataset

**3 optimum guesses for slope (m) and intercept(c)**

The scatter plot obtained was visualized from which more suitable value for intercept (c) was observed to be **20 units** and optimum slope calculated came to be **1.6** as shown below.

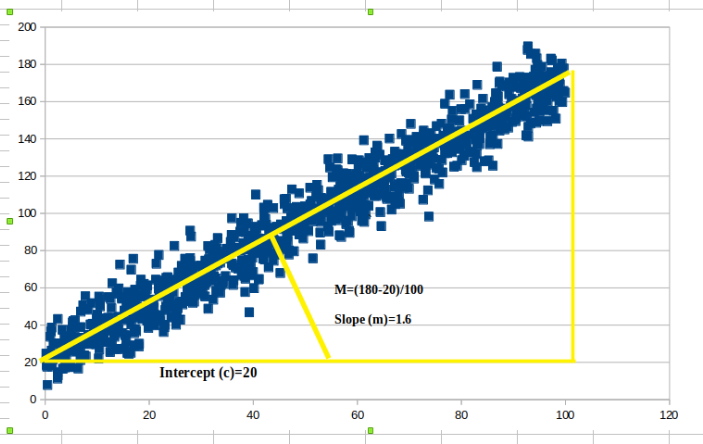
****

Figure : optimum slope and intercept visualization

Thus, one of the optimum guesses for slope and intercept is

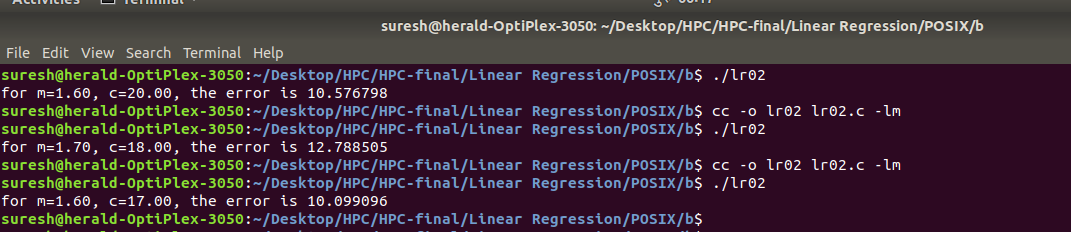
**m = 1.6 , c = 20**

Other two guesses are

**m1 = 1.7 , c1 = 18**

**m2 = 1.6 , c 2 = 17**

Then again the dataset on program file **lr02.c** was replaced with the dataset that came on linear regression program. Then predefined value of m and c were replaced with the combination of slope and intercept. The process was repeated for 3 combinations and each time the program was recompiled and rerun. The following output was obtained.



Then algorithm specified on **line-plot-02.c** program was implemented on the jupyter notebook to plot the line having slope and intercept specified above. The lines represented by those 3 different combinations were overlaid the given dataset. The output obtained was as follows:

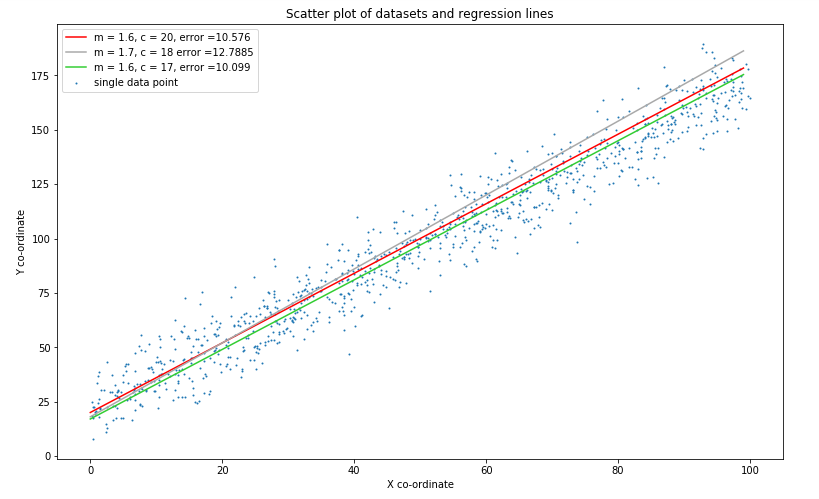


Figure : 3 regression lines plotted over dataset

**c) Running sequential version with extraneous printf statement – Linear Regression program**

The program finds minimum RMS error for given the dataset with respect to a particular combination of slope (m) and intercept (c). The program uses gradient search technique for different m and c combination on 4 different quadrants of a co-ordinate space. A base value for m and c is defined at compile time and search for finding m and c progresses until minimum error is found.

[ Screen shot left with print statement ]

The program gave m=1.52 c=20.88 and e=9.738

**Output**

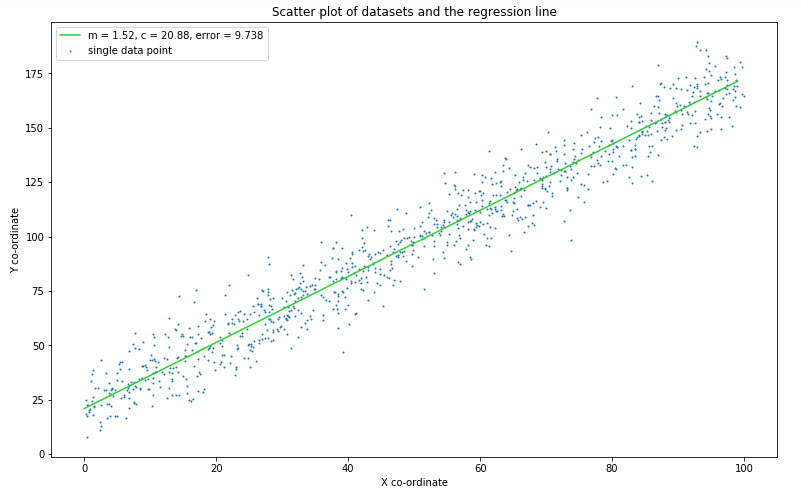
****

Figure : Line of regression generated by the program laid over dataset

**c) Running sequential version without extraneous printf statement – Linear Regression program**

The printf statement for printing intermediate m, c and error was commented then recompiled and rerun for 10 times. Initial log time log was exported to the file ‘lr\_coursework\_SEQ\_10times.CSV’. Then only time was exported and mean time was calculated on file ‘lr\_SEQ\_meanTime.CSV’. The output seen in terminal is presented below.

I/O operations inhibit parallelism because I/O operation requires more CPU time than other operations. Therefore, removal of extraneous print statement results in improved program execution time.

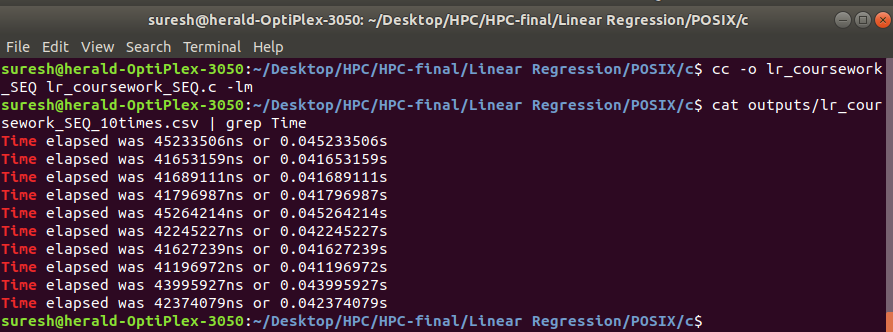


Figure : Log of running sequential linear regression program 10 times

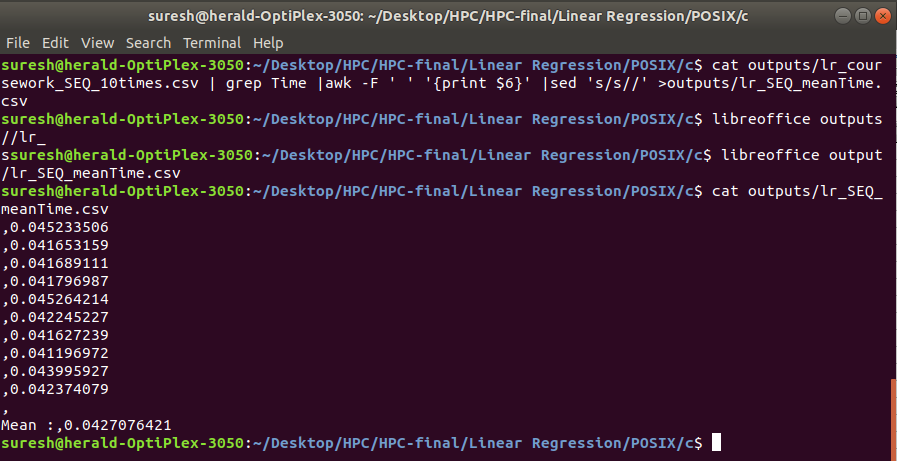


Figure : Mean execution time for sequential linear regression program

The mean execution time for sequential version of linear regression program (without extraneous print statement) was **0.0427 seconds or 42.7 milliseconds.**

**Analysis – Estimated m and c VS Actual m and c**

It is seen that the guesses for the slope m and intercept c resulted error that was **close enough** with the actual error generated. It is because human brain and eyes are incapable of visualizing distance and slope at microscopic level by just visualizing distribution of data-points on the graph. The only thing we can do is to make approximate guesses and the same thing happened in this case. On the other hand the computer has scanned numerous m, c values and calculated error for those combinations. A separate algorithm in the form of **cost function or loss function** is employed to calculate minimum error. The algorithm is well experimented and the proven one. Therefore, with the use of such algorithm the computer definitely can produce the exact values for m and c producing minimum error for the given dataset.

**Multithread version – Linear regression**

The linear regression program finds residual error for 1000 data points. Apparently, if task of calculating residual error for individual data points is divided among the threads then it would result fast computation of RMS. Therefore, this is the **hotspot** for the linear regression program. Better implementation of logic for dividing the tasks of calculating residual error for the data points can positively improve the overall program execution time.

**Data partitioning –domain decomposition into blocks**

The program is designed to have 8 pthreads. Each pthreads will process 1000/8=125 data points repeatedly and obtain sum of their residual error. Each thread will process data blocks consisting of 125 data points as shown below.

|  |  |  |
| --- | --- | --- |
| **Thread** | **Data block** | **Error calculation** |
| 1 | Data[0] – data[124] | er\_sum[0] |
| 2 | Data[125]-data[249] | er\_sum[1] |
| 3 | Data[250]-data[374] | er\_sum[2] |
| 4 | Data[375]-data[499] | er\_sum[3] |
| 5 | Data[500]-data[624] | er\_sum[4] |
| 6 | Data[625]-data[749] | er\_sum[5] |
| 7 | Data[750]-data[874] | er\_sum[6] |
| 8 | Data[875]-data[999] | er\_sum[7] |

**Coding part- multithread version 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 thread 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[] = {  {72.79,133.63},{65.97,108.84},{73.57,112.37},{68.44,116.26},  {69.77,134.42},{82.26,150.74},{90.52,151.45},{65.54,112.73},  {78.00,141.51},{65.01,115.87},{78.24,137.01},{89.72,168.27},  { 6.41,29.29},{41.94,78.67},{87.57,150.57},{49.92,103.74},  {88.08,149.06},{23.10,48.11},{80.16,142.92},{68.16,131.55},  {79.65,149.88},{34.86,75.67},{33.05,81.21},{39.53,66.68},  { 0.44,22.55},{ 4.58,29.60},{43.33,89.81},{14.75,55.18},  {69.98,132.18},{ 3.77,23.06},{89.23,170.21},{92.04,155.78},  {97.71,165.90},{94.65,179.68},{53.06,105.93},{17.17,43.75},  { 5.98,27.26},{31.88,74.13},{16.56,40.43},{ 2.42,14.63},  {67.98,115.60},{14.89,35.72},{18.70,43.34},{12.62,53.98},  {98.80,171.84},{68.10,126.07},{50.61,90.46},{83.49,140.37},  {41.04,64.55},{ 9.50,38.55},{22.79,36.30},{28.66,52.62},  {81.64,158.52},{97.34,167.65},{22.71,59.57},{90.31,160.42},  {44.79,82.58},{ 3.04,29.45},{13.90,27.06},{23.21,49.92},  { 7.44,30.07},{69.49,127.49},{37.82,74.45},{54.54,90.22},  {10.89,40.22},{29.74,59.02},{23.06,40.16},{32.07,75.67},  {58.53,115.82},{14.04,44.07},{95.08,164.71},{74.31,137.20},  {14.15,37.72},{65.91,125.14},{66.17,122.21},{19.66,61.27},  {80.70,141.25},{36.43,87.08},{71.67,134.96},{86.98,137.30},  {55.33,101.13},{71.58,135.54},{72.84,129.92},{69.30,123.43},  {42.64,81.40},{37.95,74.53},{71.42,139.15},{70.66,138.09},  { 5.43,42.19},{62.18,113.61},{27.88,90.68},{39.87,84.92},  {52.25,110.86},{91.41,167.32},{83.48,137.24},{49.62,95.93},  {30.20,63.51},{73.10,127.17},{43.01,86.17},{48.41,99.41},  {80.77,141.15},{39.91,83.57},{28.59,60.58},{87.39,170.70},  { 5.62,25.72},{28.61,57.15},{ 6.80,21.20},{19.28,46.87},  {51.43,95.72},{32.88,58.30},{29.42,62.16},{17.39,37.84},  {32.37,71.52},{18.82,48.16},{32.70,83.04},{98.59,167.04},  {17.29,59.01},{54.47,107.45},{33.16,86.70},{63.63,109.30},  {94.70,176.10},{72.66,130.00},{38.09,68.30},{59.85,121.12},  {15.20,47.75},{67.15,125.68},{34.49,77.89},{12.79,49.01},  {63.20,114.17},{88.31,145.08},{94.08,154.06},{17.13,35.64},  {78.14,150.37},{34.69,66.26},{63.10,123.38},{70.62,131.17},  {30.74,55.36},{99.95,164.83},{47.52,91.44},{75.69,123.92},  { 9.91,43.25},{55.02,97.42},{97.46,172.11},{73.10,121.40},  {47.85,79.63},{98.65,164.01},{68.22,105.17},{55.89,108.94},  {31.82,67.06},{88.28,155.71},{12.48,43.62},{26.80,67.85},  {61.11,112.87},{46.55,102.66},{ 8.26,34.68},{40.95,83.26},  {28.07,87.53},{58.18,103.27},{18.08,29.97},{70.90,120.05},  { 0.38,17.62},{64.02,115.67},{12.90,37.53},{87.97,152.75},  {56.93,87.35},{94.71,159.25},{21.24,45.39},{23.90,54.26},  {25.62,68.19},{42.05,103.04},{41.15,82.08},{95.05,162.18},  {50.95,100.52},{16.76,50.93},{38.19,84.01},{17.52,36.71},  {82.86,153.15},{ 7.75,55.47},{61.84,124.29},{95.60,166.87},  {56.43,107.41},{64.76,109.81},{80.66,141.78},{28.07,65.17},  {85.84,147.10},{45.99,104.96},{99.66,165.66},{46.66,85.33},  {73.13,122.14},{72.77,144.42},{74.04,127.51},{63.18,125.56},  { 3.23,29.36},{13.29,37.13},{30.92,67.55},{54.04,104.23},  {92.88,141.23},{18.08,45.06},{28.09,62.22},{40.27,89.91},  {28.25,68.07},{ 5.73,33.58},{21.80,59.55},{97.21,165.76},  {61.32,109.72},{48.07,98.25},{ 7.79,30.88},{42.43,86.44},  {77.40,145.35},{94.38,148.66},{77.32,132.36},{ 2.46,43.25},  {58.74,108.22},{66.30,108.14},{43.11,91.74},{92.59,172.88},  {89.70,157.06},{14.36,27.22},{ 3.99,27.66},{24.79,44.19},  {94.95,149.56},{ 7.52,23.71},{62.44,111.89},{81.65,132.88},  {92.88,169.11},{97.34,169.49},{41.69,85.09},{21.29,64.39},  {39.00,94.60},{ 4.28,26.88},{28.03,68.69},{26.23,71.77},  {66.20,140.20},{60.24,113.95},{79.97,144.34},{13.36,27.16},  {10.38,55.26},{92.55,157.68},{92.93,168.16},{25.20,49.77},  {84.16,161.49},{45.97,107.56},{48.31,103.43},{76.86,132.30},  {23.30,65.35},{22.40,60.41},{16.11,24.58},{52.97,83.19},  {58.99,103.54},{84.49,153.12},{82.50,127.49},{54.50,104.39},  {38.45,70.45},{ 3.79,27.91},{40.17,59.82},{59.19,112.19},  { 6.64,30.54},{22.42,61.99},{72.66,107.30},{39.08,85.51},  {71.39,141.06},{32.75,78.42},{25.19,64.25},{49.81,93.95},  {62.31,112.76},{60.34,128.46},{56.91,108.35},{37.47,90.16},  {80.49,131.46},{60.66,127.37},{42.28,93.38},{65.15,120.19},  {10.53,43.64},{94.26,164.08},{32.85,72.10},{ 4.08,29.55},  {28.49,66.51},{44.07,92.16},{ 0.47, 7.79},{16.56,69.72},  {43.99,83.49},{33.40,60.17},{58.99,128.97},{ 9.65,29.84},  {82.80,139.88},{74.19,135.48},{53.05,97.94},{ 4.24,30.49},  {93.34,185.59},{69.84,121.95},{38.12,83.91},{ 7.09,33.19},  {27.79,73.03},{57.41,121.47},{34.35,62.09},{85.94,156.97},  {26.90,75.62},{91.72,161.70},{72.77,138.07},{77.30,143.62},  {37.70,94.95},{39.03,81.39},{96.35,173.49},{63.13,112.12},  {84.11,146.78},{ 1.02,33.73},{36.23,64.08},{61.03,99.25},  {19.00,62.14},{10.34,35.16},{20.62,51.06},{84.02,155.13},  {53.77,101.40},{94.49,170.55},{15.31,45.26},{89.66,161.43},  {47.58,90.64},{61.27,139.24},{49.25,99.56},{18.69,56.20},  {74.17,143.02},{86.63,164.49},{24.25,44.27},{27.10,75.06},  {29.52,63.02},{26.96,52.84},{68.62,131.83},{88.67,154.58},  {39.60,87.88},{92.79,189.61},{59.38,104.30},{31.33,82.48},  {40.54,87.71},{96.84,163.83},{20.80,45.73},{63.28,109.41},  {26.42,51.11},{24.90,50.69},{ 4.89,34.05},{66.46,126.10},  {70.53,125.25},{57.88,112.00},{52.10,108.99},{68.52,142.58},  {49.91,99.99},{40.53,92.74},{26.85,68.07},{47.71,102.72},  {14.15,59.67},{45.83,81.29},{46.47,87.56},{42.95,71.00},  {82.16,133.00},{ 3.38,23.01},{54.45,129.03},{35.65,84.59},  {57.88,110.25},{61.98,118.96},{16.30,53.87},{58.46,111.68},  {16.95,75.60},{49.63,86.49},{65.71,107.71},{88.00,152.50},  {51.21,106.62},{12.91,62.43},{47.83,97.56},{94.20,177.05},  {86.97,164.58},{ 4.90,17.45},{34.91,81.03},{83.56,144.85},  {13.39,39.51},{16.98,50.93},{65.17,117.63},{64.62,93.13},  {96.57,172.56},{94.38,166.17},{63.49,110.40},{47.24,97.93},  {54.48,106.64},{82.50,144.59},{96.06,172.02},{42.78,104.61},  {26.94,55.02},{62.46,119.10},{25.75,52.04},{22.19,44.93},  { 0.91,20.64},{97.22,183.06},{62.18,104.02},{23.45,61.36},  {91.49,156.55},{48.88,99.11},{42.00,81.11},{92.47,161.80},  {95.56,178.64},{88.15,155.48},{96.40,161.35},{55.80,110.64},  {34.95,75.89},{70.68,132.51},{19.56,48.90},{30.93,63.27},  { 1.13,36.61},{17.18,47.20},{24.71,57.38},{87.36,144.44},  {31.86,70.40},{86.44,152.36},{82.95,131.84},{86.09,141.70},  {73.76,98.25},{ 8.72,40.30},{70.09,120.86},{51.16,100.44},  { 1.26,26.29},{10.41,37.41},{55.22,119.37},{53.06,108.34},  {14.20,44.40},{15.63,50.51},{69.38,126.21},{69.12,126.60},  {37.68,94.88},{24.26,64.22},{27.41,65.48},{66.61,102.05},  {13.97,56.86},{56.60,119.93},{92.21,166.29},{58.18,106.64},  {65.30,128.46},{55.01,95.98},{43.70,87.54},{91.76,160.75},  {30.79,74.59},{58.64,107.12},{27.33,59.84},{ 9.17,40.73},  {91.39,157.17},{ 1.33,17.73},{98.50,178.07},{33.80,61.56},  {80.12,138.99},{28.19,71.11},{30.96,62.95},{10.51,53.58},  {29.41,56.54},{42.18,84.03},{94.40,163.46},{92.60,152.29},  {83.49,140.39},{73.62,132.46},{42.50,93.81},{29.20,57.65},  {15.37,44.99},{15.60,31.46},{43.94,74.58},{17.18,29.07},  {87.80,168.78},{18.37,61.21},{57.04,118.97},{86.04,125.57},  {62.62,111.77},{95.05,154.71},{73.82,131.07},{89.02,146.01},  {84.55,147.22},{39.55,72.10},{76.28,128.65},{67.23,133.12},  {41.68,79.83},{ 2.47,13.05},{13.34,33.76},{36.61,70.84},  {77.91,133.83},{88.00,156.20},{68.03,110.03},{95.85,156.31},  {84.44,127.93},{19.11,49.52},{77.77,135.22},{58.21,118.17},  {94.48,156.24},{96.54,149.48},{97.17,173.39},{56.74,100.53},  {61.42,116.37},{85.17,150.99},{10.57,50.02},{47.43,112.86},  {59.82,104.72},{80.76,156.04},{10.04,45.54},{20.58,53.71},  {52.52,112.40},{40.97,78.93},{55.33,106.90},{ 0.75,19.10},  {56.11,110.27},{78.35,143.83},{51.13,98.66},{ 0.55,22.38},  {33.68,57.40},{69.95,113.22},{49.81,97.35},{40.96,86.56},  {82.85,139.64},{62.30,129.70},{33.68,73.54},{ 7.61,31.09},  { 6.37,16.63},{37.19,81.41},{18.03,28.55},{ 5.34,39.28},  {75.22,124.29},{56.26,123.11},{ 1.46,21.49},{80.08,128.54},  {56.31,97.75},{14.73,54.18},{87.13,156.92},{83.07,169.01},  {12.10,43.92},{56.99,99.14},{65.57,122.07},{67.18,114.44},  {23.93,55.51},{72.19,130.41},{55.61,109.94},{89.18,163.93},  {20.92,49.49},{98.45,169.21},{ 4.70,17.52},{53.47,105.60},  {87.04,162.61},{43.74,88.02},{78.19,132.85},{96.22,160.88},  {50.95,104.80},{30.72,63.85},{44.49,92.77},{63.43,129.91},  {35.59,80.47},{83.05,136.55},{65.14,115.55},{85.56,141.45},  {37.10,70.33},{41.11,64.70},{51.79,101.64},{76.63,137.64},  {93.31,148.28},{79.42,134.96},{25.17,40.20},{54.35,92.98},  {74.07,140.21},{56.58,104.13},{ 6.63,32.18},{30.70,61.47},  {11.11,42.11},{84.92,146.74},{63.87,117.01},{60.21,125.12},  {99.41,159.77},{69.53,115.93},{58.52,107.08},{92.45,141.85},  { 6.84,38.88},{60.57,101.11},{70.53,129.67},{92.40,168.67},  { 3.34,37.38},{53.44,104.47},{35.83,88.44},{38.21,80.65},  {58.43,90.61},{27.90,75.98},{38.48,72.74},{37.09,81.52},  {65.40,121.63},{54.03,102.55},{96.61,160.09},{91.84,169.88},  { 6.71,28.32},{35.90,73.08},{93.36,167.00},{79.48,138.17},  {19.85,38.47},{86.41,159.82},{73.77,126.20},{33.46,74.88},  {56.54,109.60},{ 5.75,42.44},{52.27,115.29},{24.08,60.94},  {79.60,142.19},{23.57,65.70},{13.91,40.15},{17.64,53.18},  {54.96,111.56},{49.65,96.22},{74.51,133.63},{52.99,95.42},  {46.33,84.66},{71.22,129.18},{49.71,91.83},{33.15,79.45},  {45.28,94.02},{93.90,173.36},{84.00,146.31},{51.50,102.51},  {56.60,88.16},{46.32,107.83},{ 3.42,16.54},{21.96,55.54},  {58.23,96.02},{ 2.59,22.52},{28.37,51.98},{38.89,81.07},  {21.83,60.13},{ 8.94,51.72},{86.64,152.78},{17.17,54.99},  {64.32,131.49},{58.10,121.18},{46.20,80.84},{56.19,121.72},  {75.21,138.59},{36.15,72.24},{79.94,155.91},{ 7.68,48.69},  {89.80,159.95},{69.94,127.85},{85.47,137.10},{44.87,85.13},  {10.30,22.08},{37.77,66.57},{80.13,140.30},{94.78,158.11},  {76.39,121.93},{60.11,119.31},{63.01,119.33},{76.78,136.61},  {85.48,146.64},{44.31,78.46},{51.50,99.24},{85.28,148.69},  {78.32,154.99},{45.21,67.96},{39.25,74.48},{23.58,54.34},  {15.80,24.98},{58.19,111.13},{69.35,138.97},{32.35,53.66},  {30.30,57.47},{94.56,166.24},{59.32,106.33},{76.54,141.72},  {26.30,69.04},{28.66,71.72},{83.81,156.22},{26.81,69.01},  {75.24,146.81},{32.89,65.04},{20.79,42.22},{32.88,75.96},  {31.61,67.95},{61.55,104.43},{18.89,48.57},{96.16,167.48},  { 0.31,18.47},{73.11,122.34},{12.10,36.86},{22.96,38.88},  {86.94,159.41},{31.14,73.21},{12.86,30.15},{29.94,71.92},  {35.00,77.54},{43.87,102.89},{43.06,91.04},{74.87,118.14},  { 1.10,24.19},{48.82,110.77},{33.18,73.13},{ 3.73,32.55},  {70.42,123.02},{61.61,99.46},{39.99,68.47},{42.83,78.92},  {77.91,145.69},{13.06,39.33},{ 7.50,23.63},{43.46,90.47},  {42.93,84.57},{ 2.03,30.25},{60.49,105.80},{15.53,27.88},  {54.74,124.56},{70.65,138.03},{25.10,47.64},{94.26,185.75},  {38.17,97.40},{92.80,170.02},{21.20,39.85},{67.59,114.41},  {64.11,120.63},{83.76,145.77},{19.35,55.64},{36.16,61.56},  {90.37,167.90},{21.39,72.92},{58.29,110.80},{42.35,96.94},  {52.76,100.77},{84.51,147.92},{90.59,156.33},{ 5.29,41.39},  {10.43,32.77},{13.27,34.56},{69.39,124.12},{42.12,95.07},  {67.27,114.34},{66.78,129.16},{56.73,114.36},{83.43,142.03},  {13.73,57.56},{32.56,76.86},{63.88,135.52},{26.32,68.97},  {31.65,80.33},{74.77,135.40},{91.76,152.01},{87.34,154.03},  {56.42,112.76},{48.37,90.82},{60.00,106.66},{42.12,99.21},  {39.87,76.32},{19.58,59.45},{97.72,155.11},{57.40,104.91},  {31.85,73.38},{25.21,50.59},{52.19,102.28},{65.35,122.77},  {90.51,158.17},{22.08,42.05},{ 9.71,34.61},{66.26,118.73},  {90.45,149.53},{32.33,76.25},{16.48,25.16},{56.03,123.63},  {94.49,183.19},{ 9.22,33.79},{76.84,158.84},{79.79,143.77},  {35.92,97.34},{84.01,152.72},{ 7.46,33.16},{10.43,27.05},  {30.50,62.71},{ 8.31,48.23},{ 7.84,23.12},{ 6.82,31.22},  {65.04,111.89},{27.99,62.28},{20.64,41.28},{56.44,105.97},  {50.93,113.77},{46.53,88.72},{47.51,91.63},{70.30,148.03},  {25.67,65.21},{41.66,93.65},{69.89,122.12},{31.33,66.22},  {99.70,177.76},{85.25,128.41},{25.35,47.29},{47.60,96.61},  {81.06,136.96},{19.06,55.30},{72.78,142.32},{30.53,62.11},  {87.93,145.74},{27.40,53.12},{53.25,107.75},{12.50,25.53},  {64.44,100.68},{19.21,58.61},{49.55,98.08},{88.11,149.89},  {11.28,33.63},{66.78,108.97},{ 9.81,29.71},{ 0.66,20.04},  {94.03,169.97},{31.36,48.78},{97.18,168.28},{67.30,104.96},  {56.85,107.49},{96.49,155.16},{27.67,56.79},{94.84,155.42},  {33.62,76.66},{40.48,110.05},{31.93,63.52},{24.85,50.02},  {85.58,150.32},{51.48,75.79},{87.54,152.34},{ 5.03,37.35},  {26.00,42.88},{22.91,63.82},{14.95,54.45},{27.91,53.19},  {63.74,117.02},{54.40,105.46},{61.65,112.88},{74.65,129.16},  {42.18,75.21},{66.59,121.36},{98.63,174.74},{89.75,149.08},  {40.37,79.59},{59.34,115.94},{39.23,46.85},{69.06,126.85},  {34.26,77.31},{43.99,85.74},{34.17,63.61},{98.96,169.19},  {69.94,139.39},{23.34,48.55},{60.89,96.00},{52.21,102.17},  {31.01,65.90},{ 4.21,29.86},{40.45,78.66},{10.80,37.54},  {20.84,52.39},{98.17,150.88},{82.98,124.83},{15.92,57.72},  {98.32,162.69},{39.66,80.52},{79.19,125.55},{88.53,153.59},  {72.29,128.45},{30.52,70.56},{56.09,96.32},{60.03,111.08},  {86.88,178.70},{19.60,42.33},{28.61,56.02},{62.44,124.67},  {48.83,88.66},{92.73,187.76},{63.44,132.76},{ 0.23,24.67},  {87.28,169.96},{37.71,79.55},{59.58,105.30},{61.78,110.97},  {53.96,100.66},{ 6.04,26.06},{38.45,57.76},{59.08,103.04},  {46.89,80.09},{38.43,73.38},{38.41,89.85},{61.34,115.62},  {84.10,152.68},{85.73,138.86},{99.31,180.46},{41.89,75.69},  {57.75,114.56},{61.00,106.40},{27.91,74.37},{44.89,77.89},  {82.90,150.13},{ 8.79,34.71},{45.39,82.27},{67.90,106.22},  {58.54,89.55},{ 1.28,38.65},{61.34,95.43},{ 4.06,22.07},  {79.91,134.58},{26.41,52.58},{ 7.32,50.59},{61.64,127.01},  { 1.49,22.00},{48.10,93.04},{70.92,118.66},{ 6.77,25.81},  {88.64,152.57},{38.34,66.50},{ 8.44,29.80},{26.53,63.70},  {80.60,130.91},{25.51,49.06},{16.28,47.94},{ 1.62,30.14},  {76.19,148.14},{11.74,37.77},{40.61,91.98},{10.29,43.36},  {39.06,74.98},{94.94,174.45},{45.83,91.54},{15.37,34.57},  {80.66,164.06},{77.15,138.65},{97.42,182.10},{28.34,61.01},  {89.49,157.67},{88.71,168.13},{56.25,129.47},{68.56,129.26},  {50.82,100.33},{15.98,53.67},{ 3.82,17.26},{66.46,127.55},  {12.21,55.00},{86.82,148.57},{48.40,93.56},{69.01,113.72},  {10.73,37.92},{92.86,146.43},{18.42,64.39},{40.56,85.90},  {72.17,125.52},{67.59,131.81},{33.09,75.00},{15.20,42.77},  { 9.32,39.95},{ 8.47,30.12},{14.41,72.50},{81.61,141.76},  {46.74,92.74},{78.90,139.34},{31.47,75.34},{59.16,119.93},  {21.87,77.64},{84.37,155.25},{58.94,98.47},{69.43,122.46},  {66.09,122.03},{47.30,90.93},{74.03,134.25},{18.83,55.97},  {46.35,95.84},{ 2.40,11.25},{91.24,173.20},{57.86,94.19},  {47.81,92.90},{64.49,122.06},{44.87,92.55},{92.13,164.69},  {71.84,131.12},{98.25,167.84},{23.27,62.17},{75.75,115.80},  {52.87,89.64},{24.59,62.05},{84.49,148.46},{63.80,136.28},  {32.45,66.16},{61.44,113.25},{ 6.85,47.24},{85.90,153.41},  {77.78,163.70},{65.64,108.72},{89.95,172.76},{46.85,77.87},  {38.88,84.07},{97.24,175.76},{62.96,125.11},{11.45,30.96},  {46.90,94.05},{10.69,42.63},{60.38,120.13},{27.31,66.44},  {78.59,125.03},{30.27,67.80},{ 8.90,40.30},{75.40,140.11},  {90.96,160.01},{29.59,58.18},{59.93,120.77},{93.30,164.66},  {24.85,82.51},{60.27,116.89},{38.49,64.89},{98.82,159.93}  }; |

**Execution time - multithread version of Linear Regression program**

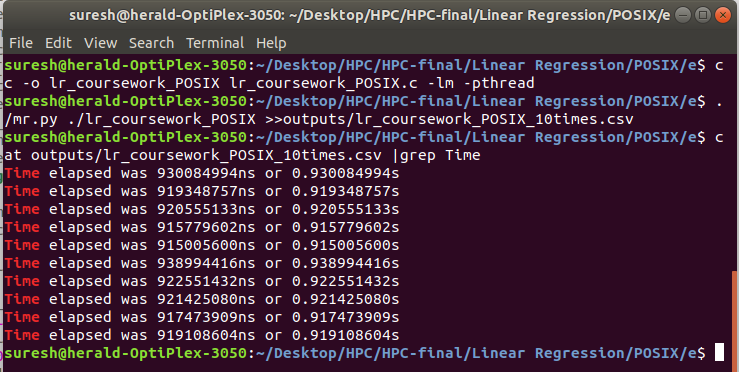


Figure : running multithread version 10 times

The output of the multithreaded program was directed to ‘lr\_coursework\_POSIX\_10times.csv’ file initially. Then only time were exported to ‘lr\_coursework\_POSIX\_meanTime.csv’ file using text utility command. The meantime was calculated using average function.

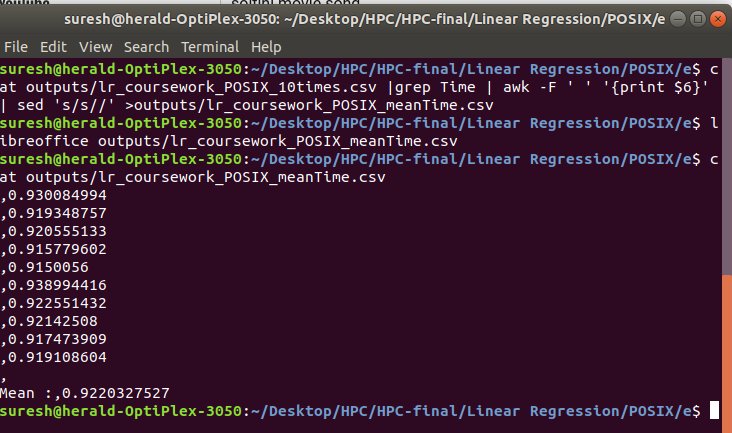


Figure : execution mean time for multithread version program

Multithreaded version of linear regression program took meantime execution of **0.922 second**.

**Execution time - sequential version of Linear Regression program**

The output of the sequntial program was directed to ‘lr\_coursework\_SEQ\_10times.csv’ file initially. Then only time were exported to ‘lr\_coursework\_SEQ\_meanTime.csv’ file using text utility command. The meantime was calculated using average function.

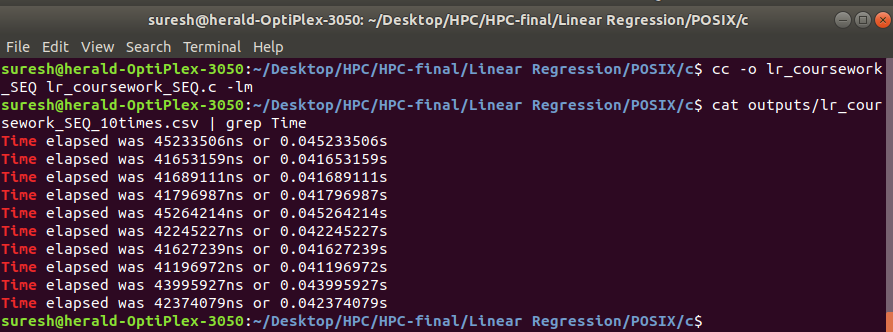


Figure : running sequential version 10 times

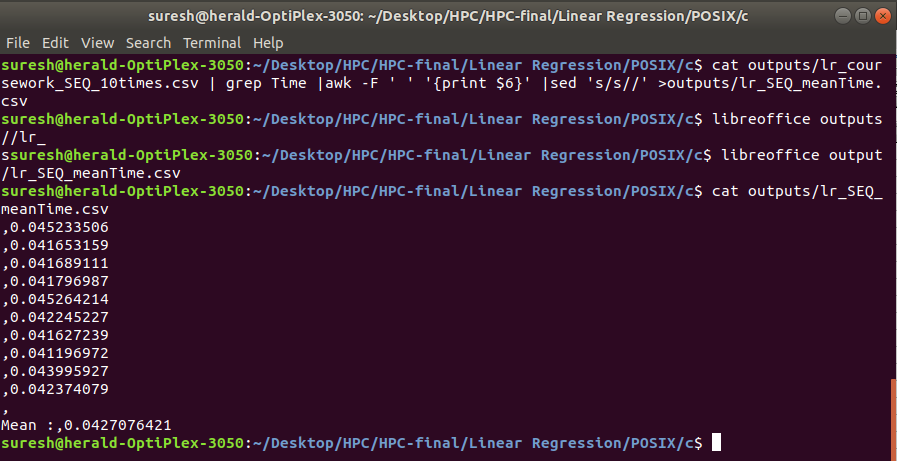


Figure : execution mean time for sequential version program

The sequential version of the program took mean time execution of **.042 second.**

**Mean execution time analysis – Sequential VS Multithreaded linear regression program.**

Mathematically, for linear regression program,

**Mean execution time Sequential version ( .042 second ) < Mean execution time Multithread Version ( .922 second )**

Although the multithread version of the linear regression program employed 8 threads to parallelize task of calculating residual error and sum of residual error and load balancing was done effectively, the mean execution time came to be almost **20 times larger** than that of sequential version. What happened is that the CPU was quick enough to process the dataset of size 1000 and intermediate calculation for error calculation. Creating threads also did the same job but with increased time cost. Threads need to be According to the program script the 8 threads are created and destroyed in each iteration in the main function. Those threads once created are scheduled and executed by the OS in random order and on top of that these threads are synchronized i.e. after residual error calculation each thread need to join to the main thread. During this process, thread completing first may need to wait for another thread which is yet to compete its job. All these steps added to the CPU overhead and which in turn resulted in increased mean execution time for the multithread version of linear regression program.

However, for sequential version of the program, the processor just needed to compute residual error and RMS sequentially. The main thread just had to deal with the dataset and error calculation part on top of that extraneous print statement was commented out. Therefore, no additional processor overhead was imposed by the program and hence the program finished with improved execution time.

# CUDA

**CUDA thread for GPUs**

Another most common thread model used for GPUs is CUDA thread model. CUDA stands for **Compute Unified Device Architecture** and it is a parallel computing platform developed by **NVIDIA Corporation** for its GPUs on graphics card. Although, primary purpose of those GPUs were to process huge 3D graphics data from videos and gaming, the developers are now using those GPUs for general purpose computing as well and this paradigm is known as **GPGPU(General Purpose computing on Graphics Processing Unit).**  NVIDIA GPUs are capable of handling data intensive applications.

(Flynn, 1966) According to Flynn, GPUs are said to have **SIMD** (**Single Instruction, Multiple Data)** architecture i.e. multiple cores in GPUs can process same instruction but with different input data at a particular instance of time. Numerous processing cores present on GPUs are slower as compared to CPU clock frequency but they support massive parallelism by running numerous concurrent threads.

CUDA thread programming is also based on C language and this API allows heterogeneous computing. Two different processing units involved in heterogeneous computing are termed as **Host** (with CPUs) and **Device** (with GPUs).

One of the common characteristics of GPUs is that it can execute many concurrent threads slowly rather than executing a single thread quickly.

**CUDA programming basics**

**\_\_host\_\_ identifier**

This identifier used with variables marks that variables are defined and stored on host memory. If used for defining functions, indicates that the function will be defined and run on host memory. Even if this identifier is not present by default variables, functions are defined and stored on host device.

**\_\_device\_\_ identifier**

This identifier used with variables or function indicates that those variables are defined and stored on graphics card while functions are run on device memory only. Creating device variables eliminates need for copying data from host machine to device machine using cudaMemcpy function and its becomes global to other device functions.

**\_\_global\_\_ identifier**

This identifier is used to define kernel function for GPUs. This identifier indicates that the kernel function must be invoked from host but will run on device memory.

**cudaMalloc()** is used to allocate memory on GPU memory. **cudaMemcpy()** function is used to copy data from host to device and vice versa. **cudaFree()** is used to free the memory allocated using cudaMalloc() earlier in the programming.

**Block**

In GPUs threads are organised into blocks. Each block can have certain number of threads. This limit is device dependent. There can be multiple blocks and these blocks can be identified or indexed using 1 dimension(x), 2 dimension (x, y) or 3 dimensions (x, y, z).

CUDA threads can also be indexed as 1D ,2D or 3D.

Dim3 data structure is used to define dimension to be used along with block and thread.

## Password Cracking

**Multithread Version using CUDA – Password Cracking**

Sequential version of password cracking program has been modified to run on CUDA GPU. Further, the program now has to resolve a set of 4 passwords each consisting of 2 Alphabet initials and 4 digits. Our requirement is to create individual threads to compute different combinations from AA to ZZ and use 4 loops to generate combination ranging from 0000 to 9999.

**Data partitioning –domain decomposition into blocks**

Since password contains 2 uppercase alphabet initials the CUDA threads need to construct combinations from AA to ZZ i.e. there will be 26\*26=676 different combinations. This also means that the program needs to create 676 different threads. To accomplish this, kernel function has to be called with **26 blocks** each consisting of **26 threads**. The kernel function can be written to manipulate **thread index** and **block index** to generate combination for 2 alphabet initials and the code for generating numeric combinations can be written there in the form of nested loops as shown in the code.

**Code for password cracking using CUDA**

|  |
| --- |
| #include <stdio.h>  #include <cuda\_runtime\_api.h>  #include <time.h>  #include <math.h>  #include <cuda.h>  #include <assert.h>  #include<stdbool.h>  #define password\_width 6  #define password\_number 4  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Compile and run with:  nvcc -o pccw\_CUDA\_object pccw\_CUDA.cu  ./pccw\_CUDA\_object    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* The passwords have been specified as array variable on device. This approach has elminated need for  using CUDA library calls such as cudaMalloc(),cudaMemcpy() etc. Futhermore, specifying passwords as array has  prevented those passwords being coppied to all the threads created since the array created has global scope\*/  //+1 is done to password with for ‘\0’ character which indicates termination of string  \_\_device\_\_ char passwordlist\_device[password\_number][password\_width+1]={"DV7611","ET6122","HR4633","SD6744"};  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);  }  /\* This function runs on device this function checks a combination of password against predefined password array.  This function returns true for password match and false otherwise. \*/  \_\_device\_\_ bool crack\_password(char \*input) {  char \*cursor\_copy;  cursor\_copy=input;  for(int i=0;i<4;i++){ //loop for selecting a password  for(int o=0;o<7;o++){ //loop for iterating through each character in a password  if (passwordlist\_device[i][o]==(char)\*input){  if(passwordlist\_device[i][o]=='\0')  {  return true;  }  input++;  }  else  break;  }    //reseting of input pointer to the beginning for next iteration  input=cursor\_copy;  }  return false;    }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This kernel function is called 26\*26 times by 26\*26 threads. infact 1 block  contains 26 threads and there are 26 such blocks. The threadIdx.x and blockIdx.x  dimensions have been used to create combinnation of two uppercase initials AA through to  ZZ. The four nested FOR loops have been used to create combinations of digits ranging  from 0000 to 9999. the array 'combination\_pswd' is used to hold such combinations  one at a time.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  \_\_global\_\_ void kernel\_function() {  char k,l,m,n;    char combination\_pswd[7];  combination\_pswd[6] = '\0';    combination\_pswd[0] = (char) blockIdx.x+65; // ASCII value of ‘A’ is 65  combination\_pswd[1] = (char) threadIdx.x+65;  for(k='0'; k<='9'; k++){  for(l='0'; l<='9'; l++){  for(m='0'; m<='9'; m++){  for(n='0'; n<='9'; n++){  combination\_pswd[2]=k;  combination\_pswd[3]=l;  combination\_pswd[4]=m;  combination\_pswd[5]=n;    if(crack\_password(combination\_pswd))  printf("Match found: %s\n",combination\_pswd);  }  }    }    }  }  int main() {  struct timespec start\_time, finish\_time;  long long int time\_elapsed;  clock\_gettime(CLOCK\_MONOTONIC, &start\_time);  kernel\_function<<<26,26>>>();  cudaThreadSynchronize();  clock\_gettime(CLOCK\_MONOTONIC, &finish\_time);  time\_difference(&start\_time, &finish\_time, &time\_elapsed);  printf("Time elapsed was %lldns or %0.9lfs\n", time\_elapsed, (time\_elapsed/1.0e9));  return 0;  } |

**Execution time - CUDA version of Password Cracking program**

The output of the file was directed to **‘pccw\_CUDA\_10times\_log.csv**’ file initially.

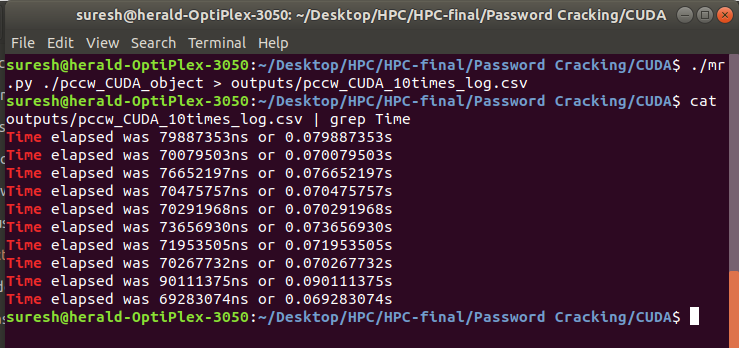


Figure : 10 times execution log for password cracking using CUDA

Then only time were exported to ‘pccw\_CUDA\_meanTime.csv’ file using text utility command. The meantime was calculated using average function.

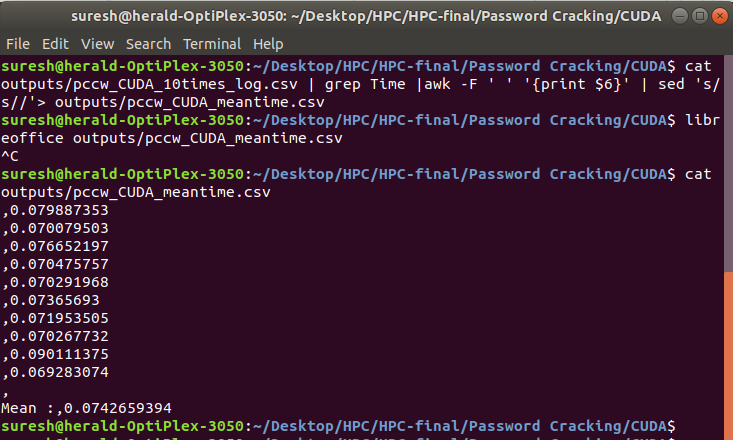


Figure : mean execution time for password cracking program using CUDA

The mean time execution for password cracking program using CUDA came to be **0.074 second.**

**Analysis – Mean execution time for sequential, POSIX thread and CUDA version of password crack**

Both sequential version and multithread version using POSIX resolved a set of 4 passwords each consisting of 2 alphabet initials and 2 digits. However, CUDA version used a set of 4 passwords which consisted of 2 alphabet initials and 4 digits. Both sequential version and POSIX thread version had to check for **26\*26\*100 =67600** different combination while resolving a password whereas, CUDA version required **26\*26\*10000=6760000** different combinations which is **100 times larger** than the combinations required for sequential version and POSIX version but still time taken by CUDA version of the program was incomparable to the time taken by the remaining version of the program.

The reason behind CUDA version taking lowest mean execution time is that,

1. The program created 26\*26=676 threads which would run in parallel each of which made combination of 2 alphabet initials. Thus, overhead of loop iteration that would be required if those threads were not used is eliminated and the number of combination that needs to be iterated for is reduced to **(6760000/676) =10000.** Thus, time required for computing 676 different combination were done at almost no time.
2. Although each thread had a loop to iterate 10000 times for generating combinations ranging from 0000 to 9999, the loop were not generating **hash** values for any of those combinations but instead **plain text comparison** took place. Since computing hash and analysing them is a lot time consuming and gives CPU overhead, this function or step was absent in CUDA version of the program.
3. Thus, **hash computation** and comparison were eliminated for the entire process of password cracking which resulted execution time to appear **in fraction of a second**.

On the other hand, POSIX and sequential version both computed and compared HASH values during cracking process which was time consuming. However, since the POSIX thread employed two threads, the task was divided into two equal halves achieving **load balancing.** Therefore, time taken by POSIX version was almost half of the time taken by sequential version. Following is diagrammatic representation of mean execution time taken by different parallel computing technology for password cracking program.

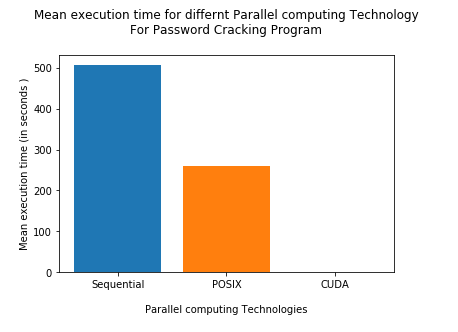


Figure : Comparison of mean execution time

## Image Processing

CUDA version of image processing program created 7200 CUDA threads to process 7200 pixels for edge detection. However, threads are grouped into blocks. 72 blocks, each containing 100 threads is created by the program. As mentioned earlier, the thread and block can have index. This index is used to identify as which thread is which and which block is which. The program sends reference to image and result to the kernel function and the kernel function computes address for each pixel using block index and thread index as follows:

**Address of pixel in device memory = blockIdx.x \* blockDim.x + threadIdx.x**

Where,

blockIdx.x= index of block

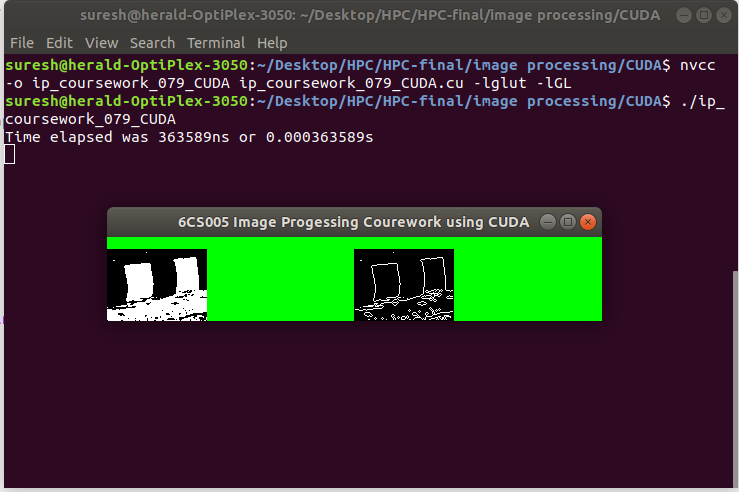
blockDim.x=dimension of block

threadIDx.x=thread index

Following is the code for the CUDA version of image processing program.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <GL/glut.h>  #include <GL/gl.h>  #include <malloc.h>  #include <signal.h>  #include <cuda\_runtime\_api.h>  #define WIDTH 100  #define HEIGHT 72  #define PIXEL\_SIZE WIDTH\*HEIGHT  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This program computes edge detection for the given image pixel value using a number  of threads that are equal with the number of pixels in the given image i.e. 7200  . Since, 72 blocks each with 100 threads are created the computation happens  quite fast.  Compile with:  nvcc -o ip\_coursework\_079\_CUDA ip\_coursework\_079\_CUDA.cu -lglut -lGL  ./ip\_coursework\_079\_CUDA > results.txt      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned char results[PIXEL\_SIZE];  unsigned char image[] = {0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,255,255,255,255,  255,255,255,0,255,255,255,255,255,255,0,0,255,255,255,255,255,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,0,  255,0,0,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,  255,255,0,0,0,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,0,0,0,255,255,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,  255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,  255,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,  255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,0,0,255,255,255,255,255,  255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,255,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,255,255,255,0,0,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,0,255,0,0,0,0,0,255,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  0,0,255,0,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,0,0,0,0,255,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,255,255,255,0,0,0,0,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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;  } |

**Program Output**



**Mean execution time calculation – Image processing CUDA version**

Initially output of the program was directed to ‘ip\_courswork\_10times.csv’ file. Then only time values in seconds were exported to ‘ip\_coursework\_meanTime.csv’ file using text utility commands.

CUDA version of the program took mean execution time of **0.0001874 second or 0.187 millisecond**.

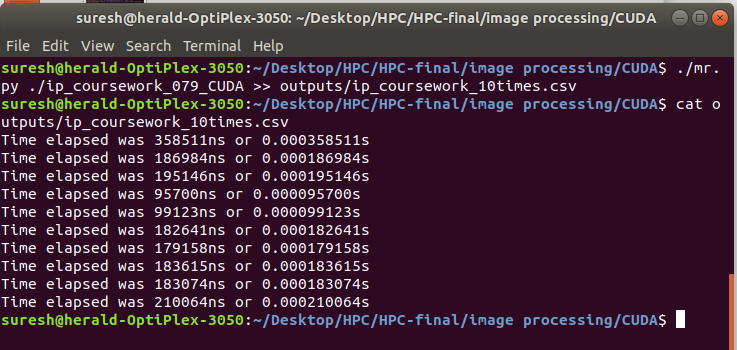
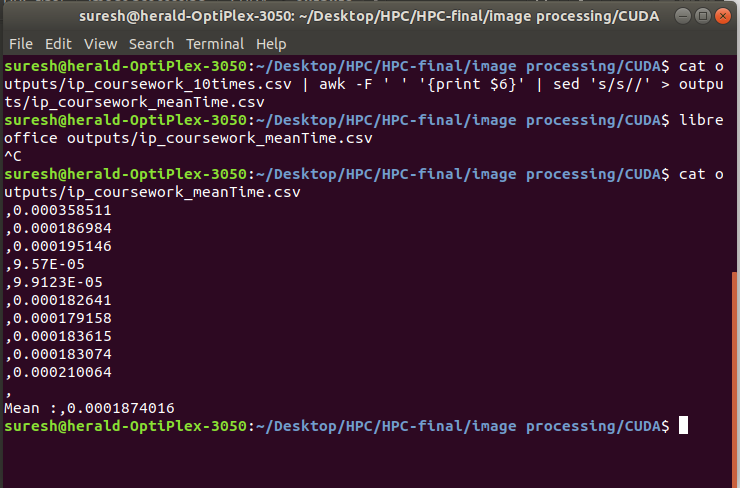
****

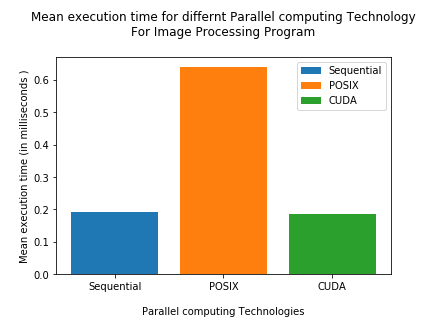
Figure : 10 times execution log



**Mean execution time comparison – sequential VS POSIX VS CUDA version**

|  |  |
| --- | --- |
| **Technology used** | **Mean execution time (in milliseconds)** |
| Sequential version | .193 |
| POSIX version | .64 |
| CUDA version | .187 |

Graphically,



According to data, mean execution time for sequential and CUDA version were nearly equal CUDA technology giving lesser mean execution time. In contrast, POSIX version took time that was almost twice of the time taken by rest of the two technologies. As it is said that, GPUs are good at processing image pixel data, the same is reflected here but this time was almost closer to the time taken by sequential version which means that CPU clock frequency is fast enough to perform equal of tasks performed by 7200 threads in GPU. However, this time difference would be considerable if number of pixels would have been larger. Further, GPUs clock speed is relatively lower than that of CPUs. Therefore, even employing 7200 threads, its working performance came to be almost equal to the CPU performance on the same node.

**Invoking more threads than number of cores**

On the other hand, POSIX version of the program employed 8 threads which would have to process pixels in loop. These threads would run on CPU cores. Since CPUs do not have many cores as GPUs have, these threads need to be executed on the same CPU cores on **time slices** provided by CPU. And mean while there are numerous processes running inside the system which needs to be dealt with too by those CPUs cores. The CPUs might have then used different thread algorithms such as FIFO (First In First Out), RR (Round Robin) and so on which means threads might need to wait for its turn to get executed. This might have created ‘**bottlenecks**’ during the program execution which in turn resulted increased program execution time as compared to other technologies used.

## Linear Regression

CUDA version of linear regression program needs to have 1000 CUDA threads for processing each data points in the given dataset to calculate error sum for particular combination of m and c. Since, this is minimisation problem that minimizes by hit and trial methods, the program needs to compute residual error for different combinations of m and c which means that the threads need to be created and RMS error needs to be evaluated until minimum RMS value is obtained.

Following is the code for CUDA version of linear regression program.

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <time.h>  #include <unistd.h>  #include <cuda\_runtime\_api.h>  #include <errno.h>  #include <unistd.h>  #define DATA\_SIZE 1000  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This program finds minimum cost or loss for by brute forcing different values of slope 'm' and 'c' in order.  Dataset has 1000 data. The program creates 1 block consisting of 1000 threads. Each thread references a single  data and receives dm and dc and calculates residual error. The total residual error or RMS error is calculated  and compared against past residual error marked as best error.  To compile:  nvcc -o lr\_coursework\_CUDA lr\_coursework\_CUDA.cu -lm    To run:  ./lr\_coursework\_CUDA  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  typedef struct {  double x;  double y;  } point\_t;  //Given dataset  point\_t data\_set[] = {  {72.79,133.63},{65.97,108.84},{73.57,112.37},{68.44,116.26},  {69.77,134.42},{82.26,150.74},{90.52,151.45},{65.54,112.73},  {78.00,141.51},{65.01,115.87},{78.24,137.01},{89.72,168.27},  { 6.41,29.29},{41.94,78.67},{87.57,150.57},{49.92,103.74},  {88.08,149.06},{23.10,48.11},{80.16,142.92},{68.16,131.55},  {79.65,149.88},{34.86,75.67},{33.05,81.21},{39.53,66.68},  { 0.44,22.55},{ 4.58,29.60},{43.33,89.81},{14.75,55.18},  {69.98,132.18},{ 3.77,23.06},{89.23,170.21},{92.04,155.78},  {97.71,165.90},{94.65,179.68},{53.06,105.93},{17.17,43.75},  { 5.98,27.26},{31.88,74.13},{16.56,40.43},{ 2.42,14.63},  {67.98,115.60},{14.89,35.72},{18.70,43.34},{12.62,53.98},  {98.80,171.84},{68.10,126.07},{50.61,90.46},{83.49,140.37},  {41.04,64.55},{ 9.50,38.55},{22.79,36.30},{28.66,52.62},  {81.64,158.52},{97.34,167.65},{22.71,59.57},{90.31,160.42},  {44.79,82.58},{ 3.04,29.45},{13.90,27.06},{23.21,49.92},  { 7.44,30.07},{69.49,127.49},{37.82,74.45},{54.54,90.22},  {10.89,40.22},{29.74,59.02},{23.06,40.16},{32.07,75.67},  {58.53,115.82},{14.04,44.07},{95.08,164.71},{74.31,137.20},  {14.15,37.72},{65.91,125.14},{66.17,122.21},{19.66,61.27},  {80.70,141.25},{36.43,87.08},{71.67,134.96},{86.98,137.30},  {55.33,101.13},{71.58,135.54},{72.84,129.92},{69.30,123.43},  {42.64,81.40},{37.95,74.53},{71.42,139.15},{70.66,138.09},  { 5.43,42.19},{62.18,113.61},{27.88,90.68},{39.87,84.92},  {52.25,110.86},{91.41,167.32},{83.48,137.24},{49.62,95.93},  {30.20,63.51},{73.10,127.17},{43.01,86.17},{48.41,99.41},  {80.77,141.15},{39.91,83.57},{28.59,60.58},{87.39,170.70},  { 5.62,25.72},{28.61,57.15},{ 6.80,21.20},{19.28,46.87},  {51.43,95.72},{32.88,58.30},{29.42,62.16},{17.39,37.84},  {32.37,71.52},{18.82,48.16},{32.70,83.04},{98.59,167.04},  {17.29,59.01},{54.47,107.45},{33.16,86.70},{63.63,109.30},  {94.70,176.10},{72.66,130.00},{38.09,68.30},{59.85,121.12},  {15.20,47.75},{67.15,125.68},{34.49,77.89},{12.79,49.01},  {63.20,114.17},{88.31,145.08},{94.08,154.06},{17.13,35.64},  {78.14,150.37},{34.69,66.26},{63.10,123.38},{70.62,131.17},  {30.74,55.36},{99.95,164.83},{47.52,91.44},{75.69,123.92},  { 9.91,43.25},{55.02,97.42},{97.46,172.11},{73.10,121.40},  {47.85,79.63},{98.65,164.01},{68.22,105.17},{55.89,108.94},  {31.82,67.06},{88.28,155.71},{12.48,43.62},{26.80,67.85},  {61.11,112.87},{46.55,102.66},{ 8.26,34.68},{40.95,83.26},  {28.07,87.53},{58.18,103.27},{18.08,29.97},{70.90,120.05},  { 0.38,17.62},{64.02,115.67},{12.90,37.53},{87.97,152.75},  {56.93,87.35},{94.71,159.25},{21.24,45.39},{23.90,54.26},  {25.62,68.19},{42.05,103.04},{41.15,82.08},{95.05,162.18},  {50.95,100.52},{16.76,50.93},{38.19,84.01},{17.52,36.71},  {82.86,153.15},{ 7.75,55.47},{61.84,124.29},{95.60,166.87},  {56.43,107.41},{64.76,109.81},{80.66,141.78},{28.07,65.17},  {85.84,147.10},{45.99,104.96},{99.66,165.66},{46.66,85.33},  {73.13,122.14},{72.77,144.42},{74.04,127.51},{63.18,125.56},  { 3.23,29.36},{13.29,37.13},{30.92,67.55},{54.04,104.23},  {92.88,141.23},{18.08,45.06},{28.09,62.22},{40.27,89.91},  {28.25,68.07},{ 5.73,33.58},{21.80,59.55},{97.21,165.76},  {61.32,109.72},{48.07,98.25},{ 7.79,30.88},{42.43,86.44},  {77.40,145.35},{94.38,148.66},{77.32,132.36},{ 2.46,43.25},  {58.74,108.22},{66.30,108.14},{43.11,91.74},{92.59,172.88},  {89.70,157.06},{14.36,27.22},{ 3.99,27.66},{24.79,44.19},  {94.95,149.56},{ 7.52,23.71},{62.44,111.89},{81.65,132.88},  {92.88,169.11},{97.34,169.49},{41.69,85.09},{21.29,64.39},  {39.00,94.60},{ 4.28,26.88},{28.03,68.69},{26.23,71.77},  {66.20,140.20},{60.24,113.95},{79.97,144.34},{13.36,27.16},  {10.38,55.26},{92.55,157.68},{92.93,168.16},{25.20,49.77},  {84.16,161.49},{45.97,107.56},{48.31,103.43},{76.86,132.30},  {23.30,65.35},{22.40,60.41},{16.11,24.58},{52.97,83.19},  {58.99,103.54},{84.49,153.12},{82.50,127.49},{54.50,104.39},  {38.45,70.45},{ 3.79,27.91},{40.17,59.82},{59.19,112.19},  { 6.64,30.54},{22.42,61.99},{72.66,107.30},{39.08,85.51},  {71.39,141.06},{32.75,78.42},{25.19,64.25},{49.81,93.95},  {62.31,112.76},{60.34,128.46},{56.91,108.35},{37.47,90.16},  {80.49,131.46},{60.66,127.37},{42.28,93.38},{65.15,120.19},  {10.53,43.64},{94.26,164.08},{32.85,72.10},{ 4.08,29.55},  {28.49,66.51},{44.07,92.16},{ 0.47, 7.79},{16.56,69.72},  {43.99,83.49},{33.40,60.17},{58.99,128.97},{ 9.65,29.84},  {82.80,139.88},{74.19,135.48},{53.05,97.94},{ 4.24,30.49},  {93.34,185.59},{69.84,121.95},{38.12,83.91},{ 7.09,33.19},  {27.79,73.03},{57.41,121.47},{34.35,62.09},{85.94,156.97},  {26.90,75.62},{91.72,161.70},{72.77,138.07},{77.30,143.62},  {37.70,94.95},{39.03,81.39},{96.35,173.49},{63.13,112.12},  {84.11,146.78},{ 1.02,33.73},{36.23,64.08},{61.03,99.25},  {19.00,62.14},{10.34,35.16},{20.62,51.06},{84.02,155.13},  {53.77,101.40},{94.49,170.55},{15.31,45.26},{89.66,161.43},  {47.58,90.64},{61.27,139.24},{49.25,99.56},{18.69,56.20},  {74.17,143.02},{86.63,164.49},{24.25,44.27},{27.10,75.06},  {29.52,63.02},{26.96,52.84},{68.62,131.83},{88.67,154.58},  {39.60,87.88},{92.79,189.61},{59.38,104.30},{31.33,82.48},  {40.54,87.71},{96.84,163.83},{20.80,45.73},{63.28,109.41},  {26.42,51.11},{24.90,50.69},{ 4.89,34.05},{66.46,126.10},  {70.53,125.25},{57.88,112.00},{52.10,108.99},{68.52,142.58},  {49.91,99.99},{40.53,92.74},{26.85,68.07},{47.71,102.72},  {14.15,59.67},{45.83,81.29},{46.47,87.56},{42.95,71.00},  {82.16,133.00},{ 3.38,23.01},{54.45,129.03},{35.65,84.59},  {57.88,110.25},{61.98,118.96},{16.30,53.87},{58.46,111.68},  {16.95,75.60},{49.63,86.49},{65.71,107.71},{88.00,152.50},  {51.21,106.62},{12.91,62.43},{47.83,97.56},{94.20,177.05},  {86.97,164.58},{ 4.90,17.45},{34.91,81.03},{83.56,144.85},  {13.39,39.51},{16.98,50.93},{65.17,117.63},{64.62,93.13},  {96.57,172.56},{94.38,166.17},{63.49,110.40},{47.24,97.93},  {54.48,106.64},{82.50,144.59},{96.06,172.02},{42.78,104.61},  {26.94,55.02},{62.46,119.10},{25.75,52.04},{22.19,44.93},  { 0.91,20.64},{97.22,183.06},{62.18,104.02},{23.45,61.36},  {91.49,156.55},{48.88,99.11},{42.00,81.11},{92.47,161.80},  {95.56,178.64},{88.15,155.48},{96.40,161.35},{55.80,110.64},  {34.95,75.89},{70.68,132.51},{19.56,48.90},{30.93,63.27},  { 1.13,36.61},{17.18,47.20},{24.71,57.38},{87.36,144.44},  {31.86,70.40},{86.44,152.36},{82.95,131.84},{86.09,141.70},  {73.76,98.25},{ 8.72,40.30},{70.09,120.86},{51.16,100.44},  { 1.26,26.29},{10.41,37.41},{55.22,119.37},{53.06,108.34},  {14.20,44.40},{15.63,50.51},{69.38,126.21},{69.12,126.60},  {37.68,94.88},{24.26,64.22},{27.41,65.48},{66.61,102.05},  {13.97,56.86},{56.60,119.93},{92.21,166.29},{58.18,106.64},  {65.30,128.46},{55.01,95.98},{43.70,87.54},{91.76,160.75},  {30.79,74.59},{58.64,107.12},{27.33,59.84},{ 9.17,40.73},  {91.39,157.17},{ 1.33,17.73},{98.50,178.07},{33.80,61.56},  {80.12,138.99},{28.19,71.11},{30.96,62.95},{10.51,53.58},  {29.41,56.54},{42.18,84.03},{94.40,163.46},{92.60,152.29},  {83.49,140.39},{73.62,132.46},{42.50,93.81},{29.20,57.65},  {15.37,44.99},{15.60,31.46},{43.94,74.58},{17.18,29.07},  {87.80,168.78},{18.37,61.21},{57.04,118.97},{86.04,125.57},  {62.62,111.77},{95.05,154.71},{73.82,131.07},{89.02,146.01},  {84.55,147.22},{39.55,72.10},{76.28,128.65},{67.23,133.12},  {41.68,79.83},{ 2.47,13.05},{13.34,33.76},{36.61,70.84},  {77.91,133.83},{88.00,156.20},{68.03,110.03},{95.85,156.31},  {84.44,127.93},{19.11,49.52},{77.77,135.22},{58.21,118.17},  {94.48,156.24},{96.54,149.48},{97.17,173.39},{56.74,100.53},  {61.42,116.37},{85.17,150.99},{10.57,50.02},{47.43,112.86},  {59.82,104.72},{80.76,156.04},{10.04,45.54},{20.58,53.71},  {52.52,112.40},{40.97,78.93},{55.33,106.90},{ 0.75,19.10},  {56.11,110.27},{78.35,143.83},{51.13,98.66},{ 0.55,22.38},  {33.68,57.40},{69.95,113.22},{49.81,97.35},{40.96,86.56},  {82.85,139.64},{62.30,129.70},{33.68,73.54},{ 7.61,31.09},  { 6.37,16.63},{37.19,81.41},{18.03,28.55},{ 5.34,39.28},  {75.22,124.29},{56.26,123.11},{ 1.46,21.49},{80.08,128.54},  {56.31,97.75},{14.73,54.18},{87.13,156.92},{83.07,169.01},  {12.10,43.92},{56.99,99.14},{65.57,122.07},{67.18,114.44},  {23.93,55.51},{72.19,130.41},{55.61,109.94},{89.18,163.93},  {20.92,49.49},{98.45,169.21},{ 4.70,17.52},{53.47,105.60},  {87.04,162.61},{43.74,88.02},{78.19,132.85},{96.22,160.88},  {50.95,104.80},{30.72,63.85},{44.49,92.77},{63.43,129.91},  {35.59,80.47},{83.05,136.55},{65.14,115.55},{85.56,141.45},  {37.10,70.33},{41.11,64.70},{51.79,101.64},{76.63,137.64},  {93.31,148.28},{79.42,134.96},{25.17,40.20},{54.35,92.98},  {74.07,140.21},{56.58,104.13},{ 6.63,32.18},{30.70,61.47},  {11.11,42.11},{84.92,146.74},{63.87,117.01},{60.21,125.12},  {99.41,159.77},{69.53,115.93},{58.52,107.08},{92.45,141.85},  { 6.84,38.88},{60.57,101.11},{70.53,129.67},{92.40,168.67},  { 3.34,37.38},{53.44,104.47},{35.83,88.44},{38.21,80.65},  {58.43,90.61},{27.90,75.98},{38.48,72.74},{37.09,81.52},  {65.40,121.63},{54.03,102.55},{96.61,160.09},{91.84,169.88},  { 6.71,28.32},{35.90,73.08},{93.36,167.00},{79.48,138.17},  {19.85,38.47},{86.41,159.82},{73.77,126.20},{33.46,74.88},  {56.54,109.60},{ 5.75,42.44},{52.27,115.29},{24.08,60.94},  {79.60,142.19},{23.57,65.70},{13.91,40.15},{17.64,53.18},  {54.96,111.56},{49.65,96.22},{74.51,133.63},{52.99,95.42},  {46.33,84.66},{71.22,129.18},{49.71,91.83},{33.15,79.45},  {45.28,94.02},{93.90,173.36},{84.00,146.31},{51.50,102.51},  {56.60,88.16},{46.32,107.83},{ 3.42,16.54},{21.96,55.54},  {58.23,96.02},{ 2.59,22.52},{28.37,51.98},{38.89,81.07},  {21.83,60.13},{ 8.94,51.72},{86.64,152.78},{17.17,54.99},  {64.32,131.49},{58.10,121.18},{46.20,80.84},{56.19,121.72},  {75.21,138.59},{36.15,72.24},{79.94,155.91},{ 7.68,48.69},  {89.80,159.95},{69.94,127.85},{85.47,137.10},{44.87,85.13},  {10.30,22.08},{37.77,66.57},{80.13,140.30},{94.78,158.11},  {76.39,121.93},{60.11,119.31},{63.01,119.33},{76.78,136.61},  {85.48,146.64},{44.31,78.46},{51.50,99.24},{85.28,148.69},  {78.32,154.99},{45.21,67.96},{39.25,74.48},{23.58,54.34},  {15.80,24.98},{58.19,111.13},{69.35,138.97},{32.35,53.66},  {30.30,57.47},{94.56,166.24},{59.32,106.33},{76.54,141.72},  {26.30,69.04},{28.66,71.72},{83.81,156.22},{26.81,69.01},  {75.24,146.81},{32.89,65.04},{20.79,42.22},{32.88,75.96},  {31.61,67.95},{61.55,104.43},{18.89,48.57},{96.16,167.48},  { 0.31,18.47},{73.11,122.34},{12.10,36.86},{22.96,38.88},  {86.94,159.41},{31.14,73.21},{12.86,30.15},{29.94,71.92},  {35.00,77.54},{43.87,102.89},{43.06,91.04},{74.87,118.14},  { 1.10,24.19},{48.82,110.77},{33.18,73.13},{ 3.73,32.55},  {70.42,123.02},{61.61,99.46},{39.99,68.47},{42.83,78.92},  {77.91,145.69},{13.06,39.33},{ 7.50,23.63},{43.46,90.47},  {42.93,84.57},{ 2.03,30.25},{60.49,105.80},{15.53,27.88},  {54.74,124.56},{70.65,138.03},{25.10,47.64},{94.26,185.75},  {38.17,97.40},{92.80,170.02},{21.20,39.85},{67.59,114.41},  {64.11,120.63},{83.76,145.77},{19.35,55.64},{36.16,61.56},  {90.37,167.90},{21.39,72.92},{58.29,110.80},{42.35,96.94},  {52.76,100.77},{84.51,147.92},{90.59,156.33},{ 5.29,41.39},  {10.43,32.77},{13.27,34.56},{69.39,124.12},{42.12,95.07},  {67.27,114.34},{66.78,129.16},{56.73,114.36},{83.43,142.03},  {13.73,57.56},{32.56,76.86},{63.88,135.52},{26.32,68.97},  {31.65,80.33},{74.77,135.40},{91.76,152.01},{87.34,154.03},  {56.42,112.76},{48.37,90.82},{60.00,106.66},{42.12,99.21},  {39.87,76.32},{19.58,59.45},{97.72,155.11},{57.40,104.91},  {31.85,73.38},{25.21,50.59},{52.19,102.28},{65.35,122.77},  {90.51,158.17},{22.08,42.05},{ 9.71,34.61},{66.26,118.73},  {90.45,149.53},{32.33,76.25},{16.48,25.16},{56.03,123.63},  {94.49,183.19},{ 9.22,33.79},{76.84,158.84},{79.79,143.77},  {35.92,97.34},{84.01,152.72},{ 7.46,33.16},{10.43,27.05},  {30.50,62.71},{ 8.31,48.23},{ 7.84,23.12},{ 6.82,31.22},  {65.04,111.89},{27.99,62.28},{20.64,41.28},{56.44,105.97},  {50.93,113.77},{46.53,88.72},{47.51,91.63},{70.30,148.03},  {25.67,65.21},{41.66,93.65},{69.89,122.12},{31.33,66.22},  {99.70,177.76},{85.25,128.41},{25.35,47.29},{47.60,96.61},  {81.06,136.96},{19.06,55.30},{72.78,142.32},{30.53,62.11},  {87.93,145.74},{27.40,53.12},{53.25,107.75},{12.50,25.53},  {64.44,100.68},{19.21,58.61},{49.55,98.08},{88.11,149.89},  {11.28,33.63},{66.78,108.97},{ 9.81,29.71},{ 0.66,20.04},  {94.03,169.97},{31.36,48.78},{97.18,168.28},{67.30,104.96},  {56.85,107.49},{96.49,155.16},{27.67,56.79},{94.84,155.42},  {33.62,76.66},{40.48,110.05},{31.93,63.52},{24.85,50.02},  {85.58,150.32},{51.48,75.79},{87.54,152.34},{ 5.03,37.35},  {26.00,42.88},{22.91,63.82},{14.95,54.45},{27.91,53.19},  {63.74,117.02},{54.40,105.46},{61.65,112.88},{74.65,129.16},  {42.18,75.21},{66.59,121.36},{98.63,174.74},{89.75,149.08},  {40.37,79.59},{59.34,115.94},{39.23,46.85},{69.06,126.85},  {34.26,77.31},{43.99,85.74},{34.17,63.61},{98.96,169.19},  {69.94,139.39},{23.34,48.55},{60.89,96.00},{52.21,102.17},  {31.01,65.90},{ 4.21,29.86},{40.45,78.66},{10.80,37.54},  {20.84,52.39},{98.17,150.88},{82.98,124.83},{15.92,57.72},  {98.32,162.69},{39.66,80.52},{79.19,125.55},{88.53,153.59},  {72.29,128.45},{30.52,70.56},{56.09,96.32},{60.03,111.08},  {86.88,178.70},{19.60,42.33},{28.61,56.02},{62.44,124.67},  {48.83,88.66},{92.73,187.76},{63.44,132.76},{ 0.23,24.67},  {87.28,169.96},{37.71,79.55},{59.58,105.30},{61.78,110.97},  {53.96,100.66},{ 6.04,26.06},{38.45,57.76},{59.08,103.04},  {46.89,80.09},{38.43,73.38},{38.41,89.85},{61.34,115.62},  {84.10,152.68},{85.73,138.86},{99.31,180.46},{41.89,75.69},  {57.75,114.56},{61.00,106.40},{27.91,74.37},{44.89,77.89},  {82.90,150.13},{ 8.79,34.71},{45.39,82.27},{67.90,106.22},  {58.54,89.55},{ 1.28,38.65},{61.34,95.43},{ 4.06,22.07},  {79.91,134.58},{26.41,52.58},{ 7.32,50.59},{61.64,127.01},  { 1.49,22.00},{48.10,93.04},{70.92,118.66},{ 6.77,25.81},  {88.64,152.57},{38.34,66.50},{ 8.44,29.80},{26.53,63.70},  {80.60,130.91},{25.51,49.06},{16.28,47.94},{ 1.62,30.14},  {76.19,148.14},{11.74,37.77},{40.61,91.98},{10.29,43.36},  {39.06,74.98},{94.94,174.45},{45.83,91.54},{15.37,34.57},  {80.66,164.06},{77.15,138.65},{97.42,182.10},{28.34,61.01},  {89.49,157.67},{88.71,168.13},{56.25,129.47},{68.56,129.26},  {50.82,100.33},{15.98,53.67},{ 3.82,17.26},{66.46,127.55},  {12.21,55.00},{86.82,148.57},{48.40,93.56},{69.01,113.72},  {10.73,37.92},{92.86,146.43},{18.42,64.39},{40.56,85.90},  {72.17,125.52},{67.59,131.81},{33.09,75.00},{15.20,42.77},  { 9.32,39.95},{ 8.47,30.12},{14.41,72.50},{81.61,141.76},  {46.74,92.74},{78.90,139.34},{31.47,75.34},{59.16,119.93},  {21.87,77.64},{84.37,155.25},{58.94,98.47},{69.43,122.46},  {66.09,122.03},{47.30,90.93},{74.03,134.25},{18.83,55.97},  {46.35,95.84},{ 2.40,11.25},{91.24,173.20},{57.86,94.19},  {47.81,92.90},{64.49,122.06},{44.87,92.55},{92.13,164.69},  {71.84,131.12},{98.25,167.84},{23.27,62.17},{75.75,115.80},  {52.87,89.64},{24.59,62.05},{84.49,148.46},{63.80,136.28},  {32.45,66.16},{61.44,113.25},{ 6.85,47.24},{85.90,153.41},  {77.78,163.70},{65.64,108.72},{89.95,172.76},{46.85,77.87},  {38.88,84.07},{97.24,175.76},{62.96,125.11},{11.45,30.96},  {46.90,94.05},{10.69,42.63},{60.38,120.13},{27.31,66.44},  {78.59,125.03},{30.27,67.80},{ 8.90,40.30},{75.40,140.11},  {90.96,160.01},{29.59,58.18},{59.93,120.77},{93.30,164.66},  {24.85,82.51},{60.27,116.89},{38.49,64.89},{98.82,159.93}  };  \_\_host\_\_ double residual\_error(double x, double y, double m, double c) {  double e = (m \* x) + c - y;  return e \* e;  }  \_\_device\_\_ double d\_residual\_error(double x, double y, double m, double c) {  double e = (m \* x) + c - y;  return e \* e;  }  \_\_host\_\_ double rms\_error(double m, double c) {  int i;  double mean;  double error\_sum = 0;    for(i=0; i<DATA\_SIZE; i++) {  error\_sum += residual\_error(data\_set[i].x, data\_set[i].y, m, c);  }    mean = error\_sum / DATA\_SIZE;    return sqrt(mean);  }  //kernel function for the program  \_\_global\_\_ void rms\_error\_DEVICE(double \*m, double \*c, double \*error\_sum\_arr, point\_t \*data\_DEVICE) {  int data\_index = threadIdx.x;  error\_sum\_arr[data\_index] = d\_residual\_error(data\_DEVICE[data\_index].x, data\_DEVICE[data\_index].y, \*m, \*c);  }  int time\_difference(struct timespec \*start, struct timespec \*finish,  long long int \*difference) {  long long int ds = finish->tv\_sec - start->tv\_sec;  long long int dn = finish->tv\_nsec - start->tv\_nsec;  if(dn < 0 ) {  ds--;  dn += 1000000000;  }  \*difference = ds \* 1000000000 + dn;  return !(\*difference > 0);  }  int main() {  int 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;    double om[] = {0,1,1, 1, 0,-1,-1,-1};  double oc[] = {1,1,0,-1,-1,-1, 0, 1};  struct timespec start, finish;  long long int time\_elapsed;    double \*dm\_DEVICE;  double \*dc\_DEVICE;  double \*error\_sum\_array\_DEVICE;  point\_t \*data\_DEVICE;  clock\_gettime(CLOCK\_MONOTONIC, &start);  be = rms\_error(bm, bc);    //memory allocation on device using error checking  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  if(cudaMalloc(&dm\_DEVICE, (sizeof(double) \* 8)))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    if(cudaMalloc(&dc\_DEVICE, (sizeof(double) \* 8)))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  cudaFree(dm\_DEVICE);  exit(1);  }    if(cudaMalloc(&error\_sum\_array\_DEVICE, (sizeof(double) \* 1000)))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  cudaFree(dm\_DEVICE);  cudaFree(dc\_DEVICE);  exit(1);  }    if(cudaMalloc(&data\_DEVICE, sizeof(data\_set)))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  cudaFree(dm\_DEVICE);  cudaFree(dc\_DEVICE);  cudaFree(error\_sum\_array\_DEVICE);  exit(1);  }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*copying host dataset to device\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  if(cudaMemcpy(data\_DEVICE, data\_set, sizeof(data\_set), cudaMemcpyHostToDevice))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    while(!minimum\_found) {  for(i=0;i<8;i++) {  dm[i] = bm + (om[i] \* step);  dc[i] = bc + (oc[i] \* step);  }    if(cudaMemcpy(dm\_DEVICE, dm, (sizeof(double) \* 8), cudaMemcpyHostToDevice))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    if(cudaMemcpy(dc\_DEVICE, dc, (sizeof(double) \* 8), cudaMemcpyHostToDevice))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    for(i=0;i<8;i++) {    double error\_sum\_array\_HOST[1000];  double error\_sum\_total;  double error\_sum\_mean;  rms\_error\_DEVICE <<<1,1000>>>(&dm\_DEVICE[i], &dc\_DEVICE[i], error\_sum\_array\_DEVICE, data\_DEVICE);  cudaThreadSynchronize();    if(cudaMemcpy(&error\_sum\_array\_HOST, error\_sum\_array\_DEVICE, (sizeof(double) \* 1000), cudaMemcpyDeviceToHost))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    for(int j=0; j<DATA\_SIZE; j++) {  error\_sum\_total += error\_sum\_array\_HOST[j];  }  error\_sum\_mean = error\_sum\_total / DATA\_SIZE;  e[i] = sqrt(error\_sum\_mean);  if(e[i] < best\_error) {  best\_error = e[i];  best\_error\_i = i;  }  error\_sum\_total = 0;  }  if(best\_error < be) {  be = best\_error;  bm = dm[best\_error\_i];  bc = dc[best\_error\_i];  } else {  minimum\_found = 1;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*freeing device memory used during the computation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  if(cudaFree(dm\_DEVICE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }  if(cudaFree(dc\_DEVICE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }  if(cudaFree(data\_DEVICE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(1);  }    if(cudaFree(error\_sum\_array\_DEVICE))  {  printf("%s\n", cudaGetErrorString(cudaGetLastError()));  exit(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;  } |

**Program output**

**Mean execution time – Linear Regression CUDA version program**

The output from the program was saved to ‘lr\_CUDA\_10timesLog.csv’ file and then only time expressed in seconds was exported to ‘lr\_CUDA\_meanTime.csv’ file. Mean execution time was calculated inside the file ‘lr\_CUDA\_meanTime.csv’.

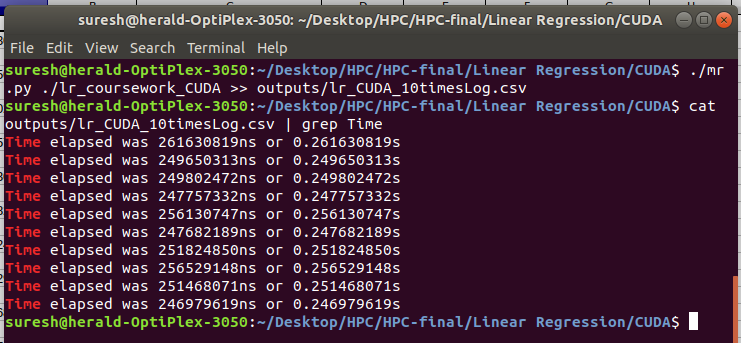


Figure : 10 times execution log for CUDA version

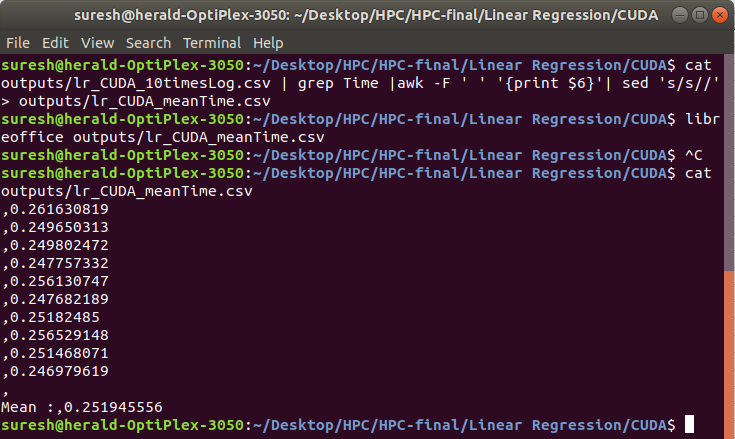


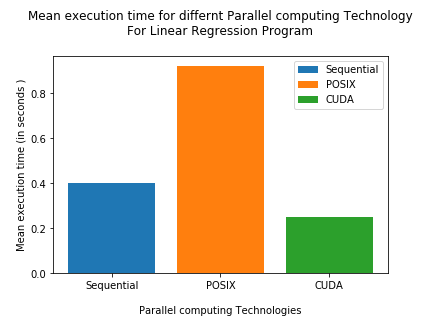
Figure : mean execution time for CUDA version

The CUDA version of linear regression program took mean execution time of **0.25 second**.

**Mean execution time comparison – sequential VS POSIX VS CUDA version of Linear regression program**

|  |  |
| --- | --- |
| **Technology used** | **Mean execution time (in seconds)** |
| Sequential version | .04 |
| POSIX version | .92 |
| CUDA version | .25 |

Graphically,



Mean execution time for CUDA version was the lowest among others. Both multithread version i.e. POSIX and CUDA program contains code for repetitive creation and synchronization of threads. Since, GPUs have numerous cores, threads created can instantly be processed by the cores but in case of POSIX threads, threads created might have to wait for their execution. Therefore, time taken by multithreaded version using POSIX has taken 4 times of the mean execution time taken by CUDA version. However, sequential version contains only loops to compute residual error, m and c space and minimum error. The CPU time required for sequential was almost half of that POSIX version.

# MPI

**Message Passing model (MPI)**

Message Passing Interface is not a library or API itself but a de-facto standard for the library that implements MPI paradigm. Message Passing Model devised by MPI Forum has become vendor independent and the standard MPI platform. The goal of MPI is to provide a standard for developing message passing programs that are practical, portable and efficient.

Parallel programming using message passing model involves movement of data from address space of one process to that of another process via co-operative communications between those processes. For example a process receiving message must have a matching sending operation that will be performed by another process.

Initially, MPI was designed for cluster computing. The nodes in cluster have their own local memory space and CPUs. The host node running a MPI program would treat those memories as distributed memory and run different processes on different nodes on the network cluster of computers. Thus, MPI supported only distributed memory architecture in the beginning but later MPI amended its library to support **shared memory architecture** as well.

MPI supports shared memory architecture by implementing distributed memory module on shared memory on a machine with shared memory.

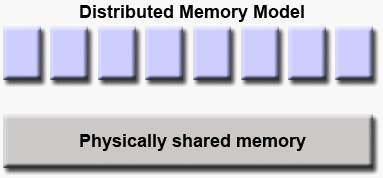
(Barney, Message Passing Interface (MPI), 2019)

Figure : Memory visualization by MPI on a single node in shared memory architecture

**MPI basics**

**Communicator:** MPI program can have a number of processes. Those processes are grouped into a object called communicator. Most of the MPI library calls requires communicator parameter to be specified. The default communicator object for an MPI program is **MPI\_COMM\_WORLD.**

**Rank:** process in an MPI program within a communicator has a rank. Ranking of processes starts from 0 and goes up to (n-1)th rank where n is the number or size of the processes for the MPI program. The rank for a process is used to identify an MPI process which is very important for the purpose of **Inter-process communication**.

**Point to point communication**

MPI processes often communicate with each other via send and receive operation. One process can send data or message to another process for example by using MPI\_Send() while another process may receive sent messages or data using MPI\_Recv() function. This type of communication is called **point to point communication** in MPI program.

**Collective communication**

MPI processes may need to send message or data collectively as well. Further, a process may need to communicate to every other processes in a communicator group or specific group of process etc. To address such requirement, MPI C based library has functions such as MPI\_Bcast(),MPI\_Scatter(), MPI\_Gather() etc. Since this, these libary calls are used to communicate to other processes as group it is called **collective communication**.

**MPI broadcast**: MPI broadcast is used to broadcast a message to all other processes belonging to a common **communicator.** if a process with a particular rank wants to broadcast message to other processes it needs to use its rank in the MPI\_Bcast () and other receiving processes must make call to same MPI\_Bcast() to receive the broadcasted message. This collective communication is useful for sending same message to numerous processes.

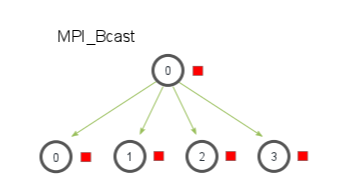
(Barney, Message Passing Interface (MPI), 2019)

Figure : MPI broadcast operation

**Blocking Operation**

Point to point communication calls in MPI can be either blocking or non-blocking. Blocking operation is a synchronous operation while non-blocking operation is asynchronous.

**MPI\_Send()** is a blocking operation because when a process sends data using this call, the call waits for acknowledgment regarding delivery of the message either to the receiver process or data buffer. Only after receiving acknowledgement, instructions following the call are executed.

Similarly, **MPI\_Recv()** is also a blocking operation because upon executing this call, the process waits for the message it expects to receive and returns only after the data or message is arrived or received.

**Structure of MPI program**

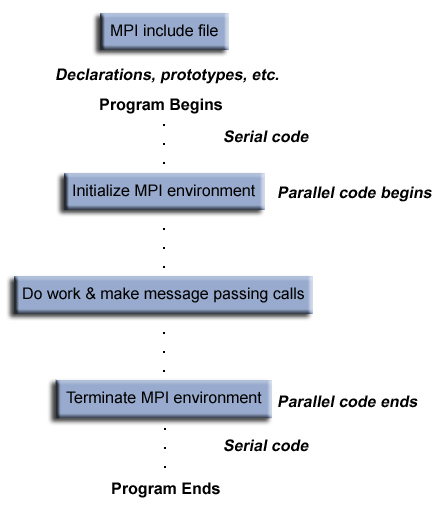
(Barney, Message Passing Interface (MPI), 2019)

Figure : General structure of an MPI program

For this module, MPI programs on a single node have been examined rather than running them on network cluster.

## Password Cracking

MPI version of password cracking program uses 3 different processes to crack a set of 4 passwords each consisting of 2 alphabet initials and 4 digit numbers. Process with rank 0 is regarded as master process while others with rank 1 and 2 are regarded as slave processes. Master process simply distributes password to crack to the slave processes via broadcast. After slave processes finishes their tasks of cracking password they send acknowledge back to master process. The master process waits for acknowledgement from the slave processes and upon receiving the acknowledgement from both processes the process shows time elapsed during password crack by two slave process and exists the program.

**Data partitioning –domain decomposition into blocks**

Just as in multithreaded version using POSIX, the task of password cracking has to be divided among two slave processes for **load balancing.**

Since password consists of 2 alphabets and 4 digits there will be **26\*26\*100\*100=6760000** combinations this can be divided among the processes as shown below:

|  |  |  |
| --- | --- | --- |
| **Process Rank** | **Number of combination** | **Dataset** |
| 1 | 6760000/2=3380000 | AA0000 –MZ9999 |
| 2 | 6760000/2=3380000 | NA0000 – ZZ9999 |

**Source code**

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <string.h>  #include <time.h>  #include <crypt.h>  #include <mpi.h>  #define NUM\_PASSWORD 4  #define HASH\_LENGTH 93  #define SIZE\_OF\_ARRAY NUM\_PASSWORD\*HASH\_LENGTH  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  The variable names and the function names of this program is same as provided by the university.  The added variable and function are the only changes made to this program.    To compile:  mpicc -o pccw\_MPI pccw\_MPI.c -lrt -lcrypt    To run 3 processes on this computer:  mpirun -n 3 ./pccw\_MPI  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    /\* those variables are made global so that each processes can access them \*/  char \*saltAndEncryptedText;  char salt[7];  char \*enc;  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 function called by one of the process to decrypt hash value supplied  the function makes comparsion brute-forcely. the function check for password starting form initial  letter 'A' to 'M' \*/  void crackA2M(char \*input\_password,int rank ){    int solution\_counter=0;  int w, x, y, z; // Loop counters  char plain[7];    substr(salt, input\_password, 0, 6);  for(w='A'; w<='M'; w++){  for(x='A'; x<='Z'; x++){  for(y=0; y<=99; y++){  for(z=0;z<=99;z++){  sprintf(plain, "%c%c%02d%02d",w, x, y, z);  enc = (char \*) crypt(plain, salt);  solution\_counter++;    if(strcmp(input\_password, enc) == 0){  printf("#%-8d%s %s (computed by Process with rank: %d)\n", solution\_counter, plain, enc,rank);  } else {  //printf("%-8d%s %s (computed by Process with rank: %d)\n", solution\_counter, plain, enc,rank);  }  }  }  }  }  printf("Total solutions tried by \nProcess with rank : %d = %d solutions explored\n",rank, solution\_counter);  }  /\*\*\* the purpose of this function is same as that of 'crackA2M' except that it checks for passwords  starting from 'N' to 'Z' \*\*\*/  void crackN2Z(char \*input\_password, int rank){  int solution\_counter=0;  int w, x, y, z; // Loop counters  char plain[7];  substr(salt, input\_password, 0, 6);  for(w='N'; w<='Z'; w++){  for(x='A'; x<='Z'; x++){  for(y=0; y<=99; y++){  for(z=0;z<=99;z++){  sprintf(plain, "%c%c%02d%02d",w, x, y, z);  enc = (char \*) crypt(plain, salt);    solution\_counter++;    if(strcmp(input\_password, enc) == 0){  printf("#%-8d%s %s (computed by Process with rank: %d)\n", solution\_counter, plain, enc,rank);  } else {  //printf("%-8d%s %s (computed by Process with rank: %d)\n", solution\_counter, plain, enc,rank);  }  }  }  }  }  printf("Total solutions tried by \nProcess with rank : %d = %d solutions explored\n",rank, solution\_counter);  }  void main(int argc, char\*\* argv) {    struct timespec start, finish;  long long int time\_elapsed;  clock\_gettime(CLOCK\_MONOTONIC, &start);  int size, rank;  int i;    MPI\_Init(NULL, NULL);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if(size != 3) {  if(rank == 0) {  printf("This program needs to run on exactly 3 processes\n");  }  } else {  if(rank ==0){    clock\_gettime(CLOCK\_MONOTONIC, &start);  int acknowledgement1,acknowledgement2;  /\*\*\*\*\* 4 hash value of password supplied \*\*\*\*\*\*\*/  char encrypted\_passwords[NUM\_PASSWORD][HASH\_LENGTH] = {  "$6$KB$dw0LU6u6RJrWAzlWY7hlO/v.XKsTqnC3U30YtDWO6Hl3Ittgo1NvIVh1n.TE7DYSKNreo4hGacw2xKxYwFuLN1",  "$6$KB$wrahZlw7Avy5j5QbqbETGENjda8H2GZupQD.uZvykuIe2Qj2jvWTJYnvetahJvPV.V8qcFH1yg72Mkm8aVia/.",  "$6$KB$.Vw6HbJVM1NEhBLjiTfHYgjmEiYKoZoF7ORm8nmG0WCqdHE6c1y5y0Eu1/csAm9mumcGkLnqehBlVAgeEOcen.",  "$6$KB$xzI9MgFMqI3f0VyGfr.kMUizYmPJyexesCylSOl9CF7awMrUIuEofzKNyYkjlmWlCKiu8iLaT.tWCBJSRy4UR0"  };  MPI\_Bcast(&encrypted\_passwords, SIZE\_OF\_ARRAY, MPI\_BYTE, 0, MPI\_COMM\_WORLD);    MPI\_Recv(&acknowledgement1, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  MPI\_Recv(&acknowledgement2, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);    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));    } else if(rank==1){    /\* variable for counting solutions examined by two processes \*/  int response;  char hashes\_copy\_1[NUM\_PASSWORD][HASH\_LENGTH];  MPI\_Bcast(&hashes\_copy\_1, SIZE\_OF\_ARRAY, MPI\_BYTE, 0, MPI\_COMM\_WORLD);    for(i=0;i<NUM\_PASSWORD;i<i++) {  crackA2M(hashes\_copy\_1[i],rank);  }  response=1;  MPI\_Send(&response, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);  }  else if(rank==2){  /\* variable for counting solutions examined by two processes \*/  int response;    char hash\_copy\_2[NUM\_PASSWORD][HASH\_LENGTH];    MPI\_Bcast(&hash\_copy\_2, SIZE\_OF\_ARRAY, MPI\_BYTE, 0, MPI\_COMM\_WORLD);    for(i=0;i<NUM\_PASSWORD;i<i++) {  crackN2Z(hash\_copy\_2[i],rank);  }  response=1;  MPI\_Send(&response, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);  }  }  MPI\_Finalize();  } |

**Output**

**Mean execution time comparison – sequential VS POSIX VS CUDA VS MPI version of Password cracking program**

|  |  |  |
| --- | --- | --- |
| **Technology used** | **Mean execution time (in minutes)** | **Password structure** |
| Sequential version | 8.4 | 2 initials ,2 digits |
| POSIX version | 4.3 | 2 initials, 2 digits |
| CUDA version | 0.00123 | 2 initials, 4 digits |
| MPI |  | 2 initials, 4 digits |

Apparently, among 4 of the technologies used for password cracking purpose, CUDA appears to be the most efficient technology as it results in extremely less execution time. The primary reason for this stellar performance is the huge number of cores the GPUs have. Since, there GPUs can run 1000 of threads parallel it can check for considerable number of combination of passwords in no time. Since, there no hashing algorithm is used; plain text comparison is quicker which adds to improved GPU performance. However, in practice, the passwords to be cracked are not in plain text form. Generally, passwords are available in the form of encrypted hash values. In that case, this technology may not work at all.

## Image Processing

MPI version of the image processing program creates 5 processes. Process with rank 0 is a root process, while processes with rank 1 to 4 are slave processes. The root process sends index value to other processes which is actually index of the pixel to begin edge detection task for those processes. The slave processes use 1800 pixels each and reads the processed pixels as sub\_result array. This sub\_result array is sent back to the root process by each of the slave processes and the root process reads those sub\_result arrays into a single result array in order.

**Data partitioning –domain decomposition into blocks**

The task of dividing pixel among 4 processes for edge detection can be visualised as follows:

Total pixels=**7200**

Number of slave processes =**4**

Pixel /process=7200/4=**1800**

|  |  |  |
| --- | --- | --- |
| **Process Rank** | **Number of pixels to process** | **Data range** |
| 1 | 1800 | Image[0]-image[1799] |
| 2 | 1800 | Image[1800]-image[3599] |
| 3 | 1800 | Image[3600]-image[1799] |
| 4 | 1800 | Image[5400]-image[7199] |

**Source Code**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <GL/glut.h>  #include <GL/gl.h>  #include <malloc.h>  #include <signal.h>  #include <mpi.h>  #include <time.h>  /\* defining image details as constants \*/  #define WIDTH 100  #define HEIGHT 72  #define IMAGE\_SIZE (WIDTH\*HEIGHT)  #define BLOCK\_SIZE (IMAGE\_SIZE/4)  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This program detects edges for the given image using MPI techniques. The program creates 5  processes. Process with rank 0 is master and others are slave process.  - Root process sends index marking the beginning of the image to begin processing for edge detection.  - Slave processes upon receiving index calls edge\_detection function and obtains processed pixel block into  a seperate block  - Slave processes sends back the processed image block back to the root process  - the root process reads the sub blocks of pixels into a single results  and displays the result  To compile adapt the code below wo match your filenames:  mpicc -o ip\_coursework\_079\_MPI ip\_coursework\_079\_MPI.c -lglut -lGL -lm  To run 5 processes on this computer:  mpirun -n 5 ./ip\_coursework\_079\_MPI  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  unsigned char results[IMAGE\_SIZE];  unsigned char image[] = {0,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,0,255,255,255,0,255,255,255,255,  255,255,255,0,255,255,255,255,255,255,0,0,255,255,255,255,255,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,0,  255,0,0,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,255,0,  255,255,0,0,0,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,  255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,255,255,255,0,0,0,255,255,0,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,255,255,255,255,0,0,0,0,255,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,  255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,  255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,  255,0,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,255,255,255,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,255,255,255,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,  255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,0,0,255,255,255,255,255,  255,255,255,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,255,255,255,255,255,0,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,255,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,255,0,0,0,0,0,0,0,0,0,255,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,255,255,255,0,0,255,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,0,255,0,0,0,0,0,255,0,0,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,0,0,0,255,255,255,  255,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  0,0,255,0,0,255,255,0,0,255,255,255,255,255,255,255,255,255,255,  255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,0,0,0,0,255,0,0,0,0,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,0,0,255,255,255,0,0,0,0,255,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  255,255,255,0,0,255,0,0,255,255,0,255,255,255,255,255,255,255,255,  0,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  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,  0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  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,  0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,  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,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  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,  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,  0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,  255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,255,255,255,255,255,0,255,255,255,255,255,255,255,255,255,255,255,  255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  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);  }  void detect\_edges(unsigned char \*in, unsigned char \*out) {  int i;  for(i=0;i<BLOCK\_SIZE;i++) {  int x, y; // the pixel of interest  int b, d, f, h; // the pixels adjacent to x,y used for the calculation  int r; // the result of calculate    y = i / WIDTH;  x = i - (WIDTH \* y);  if (x == 0 || y == 0 || x == WIDTH - 1 || y == HEIGHT - 1) {  out[i] = 0;  } else {  b = i + WIDTH;  d = i - 1;  f = i + 1;  h = i - WIDTH;  r = (in[i] \* 4) + (in[b] \* -1) + (in[d] \* -1) + (in[f] \* -1)  + (in[h] \* -1);  if (r > 0) { // if the result is positive this is an edge pixel  out[i] = 255;  } else {  out[i] = 0;  }  }  }  }  void sigint\_callback(int signal\_number){  printf("\nInterrupt from keyboard\n");  }  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:    break;  default:  printf("\nPress escape to exit\n");  break;  }  }  void main(int argc, char \*\*argv) {    struct timespec start, finish;  long long int difference;  int size, rank;  MPI\_Init(NULL, NULL);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if(size != 5) {  if(rank == 0) {  printf("This program needs 5 processes\n");  }  } else {  if(rank ==0){  int index=0;    signal(SIGINT, sigint\_callback);  clock\_gettime(CLOCK\_MONOTONIC, &start);    /\* sending of pixel-index of beginning each blocks to 4 different processes \*/  for(int i=1;i<size;i++){  MPI\_Send(&index, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);  index+=BLOCK\_SIZE;  }    index=0;    /\*receiving processed block back from each process and reading them in results array in order \*/  for(int i=1;i<size;i++){  MPI\_Recv(&results[index], BLOCK\_SIZE, MPI\_UNSIGNED\_CHAR, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  index+=BLOCK\_SIZE;  }      clock\_gettime(CLOCK\_MONOTONIC, &finish);  time\_difference(&start, &finish, &difference);  printf("Time elapsed was %lld or %0.9lf\n", difference,  difference/1.0e9);  glutInit(&argc, argv);  glutInitWindowSize(WIDTH \* 2,HEIGHT);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_LUMINANCE);    glutCreateWindow("6CS005 Image Progessing Courework using MPI");  glutDisplayFunc(display);  glutKeyboardFunc(key\_pressed);  glClearColor(0.0, 1.0, 0.0, 1.0);  glutMainLoop();  }  else {  if(rank == 1){    int starting\_index;  unsigned char sub\_result[BLOCK\_SIZE];  MPI\_Recv(&starting\_index, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);    /\* sending address of the block of image to be processed and reading them into a sub\_result block  to detect\_edges function\*/  detect\_edges(&image[starting\_index], sub\_result);  /\* sending processed image block back to the master process \*/  MPI\_Send(&sub\_result, BLOCK\_SIZE, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);  }  else if(rank == 2){    int starting\_index;  unsigned char sub\_result[BLOCK\_SIZE];  MPI\_Recv(&starting\_index, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  /\* sending address of the block of image to be processed and reading them into a sub\_result block  to detect\_edges function\*/  detect\_edges(&image[starting\_index], sub\_result);  /\* sending processed image block back to the master process \*/  MPI\_Send(&sub\_result, BLOCK\_SIZE, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);    }  else if(rank == 3){    int starting\_index;  unsigned char sub\_result[BLOCK\_SIZE];  MPI\_Recv(&starting\_index, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  /\* sending address of the block of image to be processed and reading them into a sub\_result block  to detect\_edges function\*/  detect\_edges(&image[starting\_index], sub\_result);  /\* sending processed image block back to the master process \*/  MPI\_Send(&sub\_result, BLOCK\_SIZE, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);    }  else if(rank == 4){    int starting\_index;  unsigned char sub\_result[BLOCK\_SIZE];  MPI\_Recv(&starting\_index, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  detect\_edges(&image[starting\_index], sub\_result);  MPI\_Send(&sub\_result, BLOCK\_SIZE, MPI\_UNSIGNED\_CHAR, 0, 0, MPI\_COMM\_WORLD);  }  }  }  MPI\_Finalize();    } |

**Output**

**Mean execution time comparison – sequential VS POSIX VS CUDA VS MPI version of Password cracking program**

|  |  |  |
| --- | --- | --- |
| **Technology used** | **Mean execution time (in minutes)** | **Password structure** |
| Sequential version | 8.4 | 2 initials ,2 digits |
| POSIX version | 4.3 | 2 initials, 2 digits |
| CUDA version | 0.00123 | 2 initials, 4 digits |
| MPI |  | 2 initials, 4 digits |

## Linear Regression

Paste your source code for your MPI based linear regression

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

Write a short analysis of the results

# Verbose Repository Log

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

svn update

svn –v log > log.txt

gedit log.txt

Then select, copy and paste the text here