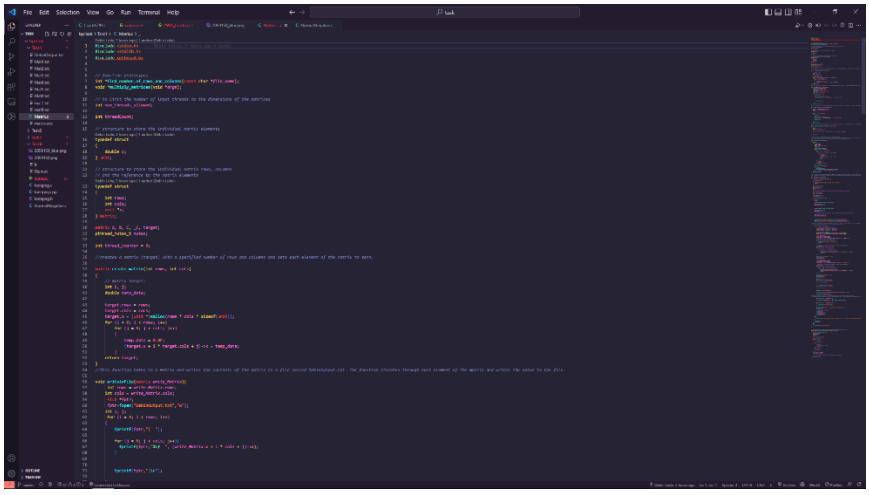
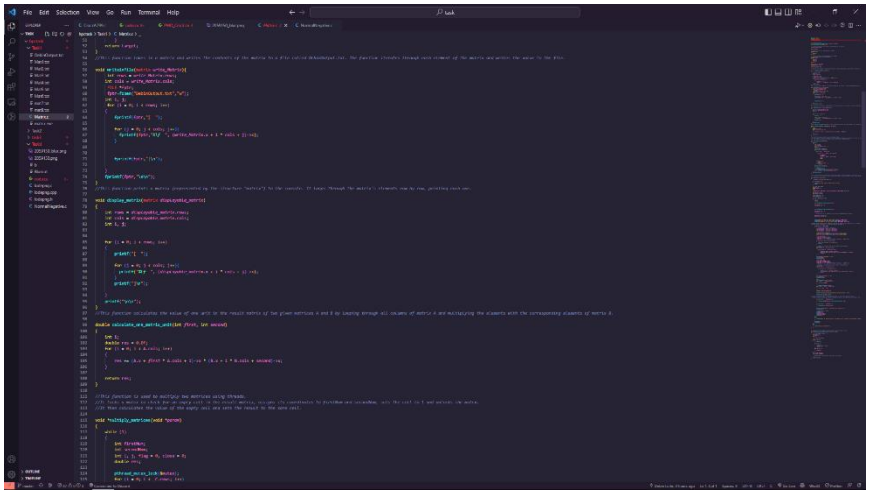
1. Matrix Multiplication (15% - 100 marks)

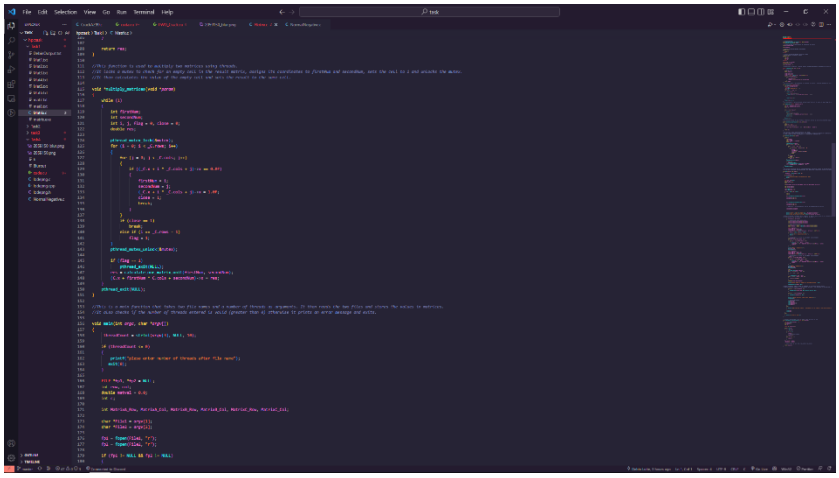
You will create a matrix multiplication program which uses multithreading, you will be taught two CPU multithreading concepts: POSIX threads and Open Multi-Processing (OpenMP). Matrices are often two-dimensional arrays varying in sizes (can also be 3D), for your application, you will only need to multiply two-dimensional ones. Your program will read in the matrix from a file (txt), store them appropriately using dynamic memory and multiply them by splitting the tasks across "n" threads (any number of threads via command line arguments). Each matrix will be in a separate file. You should also use command line arguments to allow the user to enter the which matrices need to be multiplied. The data files will be given to you however, you will need to initially identify the dimensions of the matrix to determine whether they can be multiplied. If the matrices cannot be multiplied, your program should notify the user. For example, if Matrix A is 3x3 and Matrix B is 2x2, you cannot multiply them. If Matrix A is 2x3 and Matrix B is 2x2, then this can be multiplied. You will need to research how to multiply matrices, this will also be covered in the lectures. The resulting matrix should be outputted to a file. It is up to you which multithreading library you use (Pthreads or OMP).

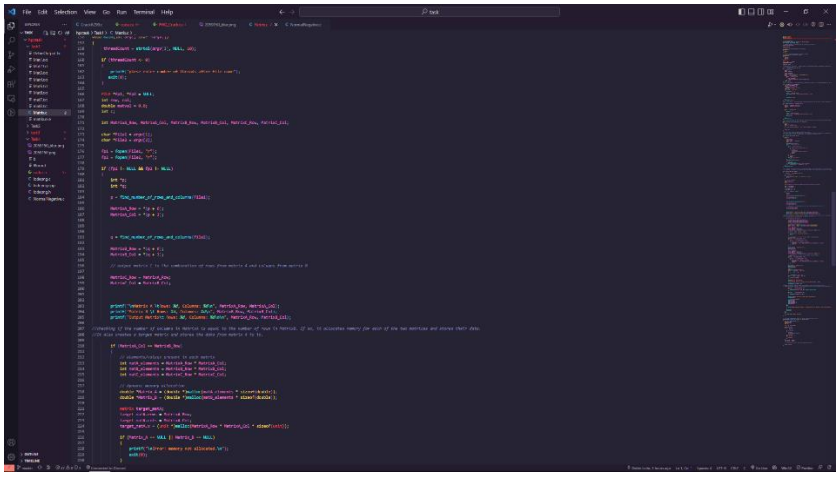
Code Screenshot and explanation

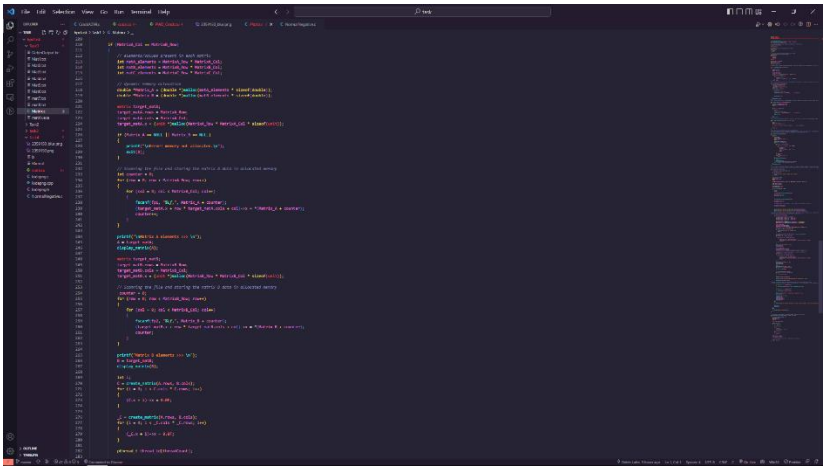


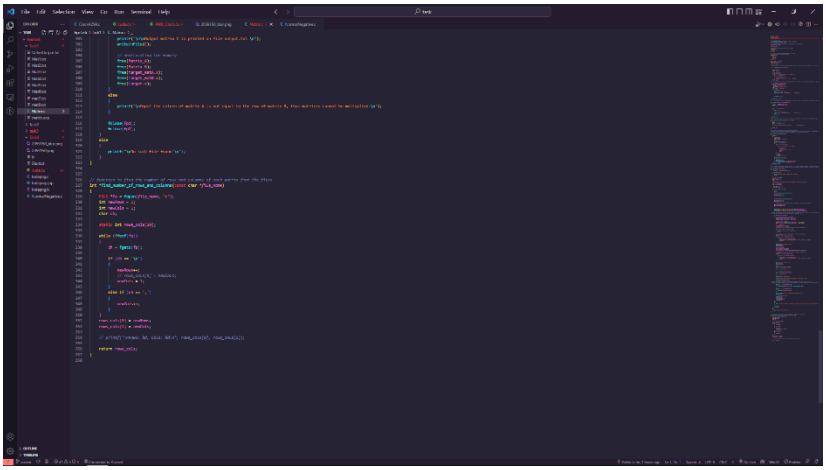
2059150_debin_luitel











Explanation in brief:

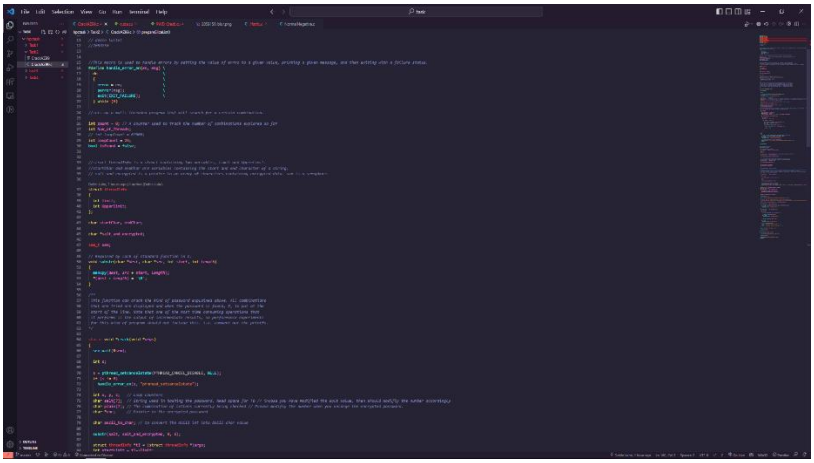
This code is used to multiply two matrices together using threads. It takes in two files containing the data for the two matrices and a number of threads as arguments. It then reads the two files, stores the data in two matrices, checks if the number of columns in MatrixA is equal to the number of rows in MatrixB, and if so, allocates memory for each of the two matrices and stores the data. It also creates a target matrix and stores the data from matrix A to it.

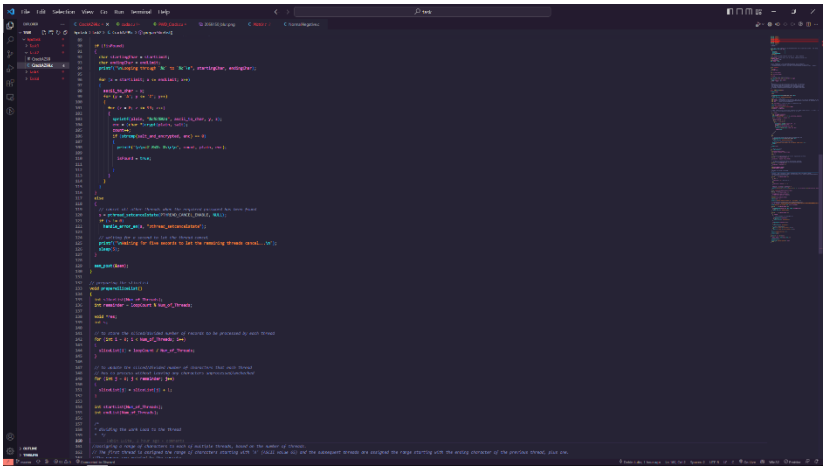
The code then passes the thread count to a function that creates "threadCount" number of threads and makes them run the "multiply_matrices" function. The "multiply_matrices" locks a mutex to check for an empty cell in the result matrix, assigns its coordinates to firstNum and secondNum, sets the cell to 1 and unlocks the mutex. It then calculates the value of the empty cell and sets the result to the same cell. Once all threads have completed, the code prints the output matrix C on a file, deallocates all the memory used, and closes the two files.

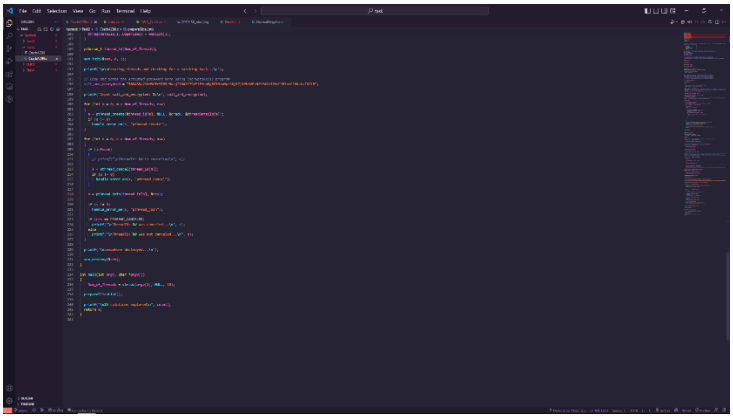
Task 2

2. Password cracking using multithreading (15% - 100 marks)

In this task, you will be asked to use the “crypt” library to decrypt a password using multithreading.You will be provided with two programs. The first program called “EncryptSHA512.c” allows you toencrypt a password. For this assessment, you will be required to decrypt a 4-character passwordconsisting of 2 capital letters, and 2 numbers. The format of the password should be“LetterLetterNumberNumber.” For example, “HP93.” Once you have generated your password, thisshould then be entered into your program to decrypt the password. The method of input for theencrypted password is up to you. The second program is a skeleton code to crack the password on asingle thread without any multithreading syntax. Your task is to use the pthread or omp library tosplit the workload over many threads and find the password. Once the password has been found,the program should finish meaning not all combinations of 2 letters and 2 numbers should beexplored unless it’s ZZ99 AND the last thread happens to finish last.







Explanation in brief

This code sets up a multi-threaded program to search for a certain combination. The macro 'handle_error_en' is used to handle errors by setting the value of errno to a given value, printing a given message, and then exiting with a failure status. A struct threadInfo is used to store two variables, limit and Upperlimit, which will be used to store the range of characters from the given encrypted data that the thread will have to crack. A semaphore is used to allow multiple threads to use the same resources. A counter is used to track the number of combinations explored so far and a boolean is used to indicate if the password has been found.

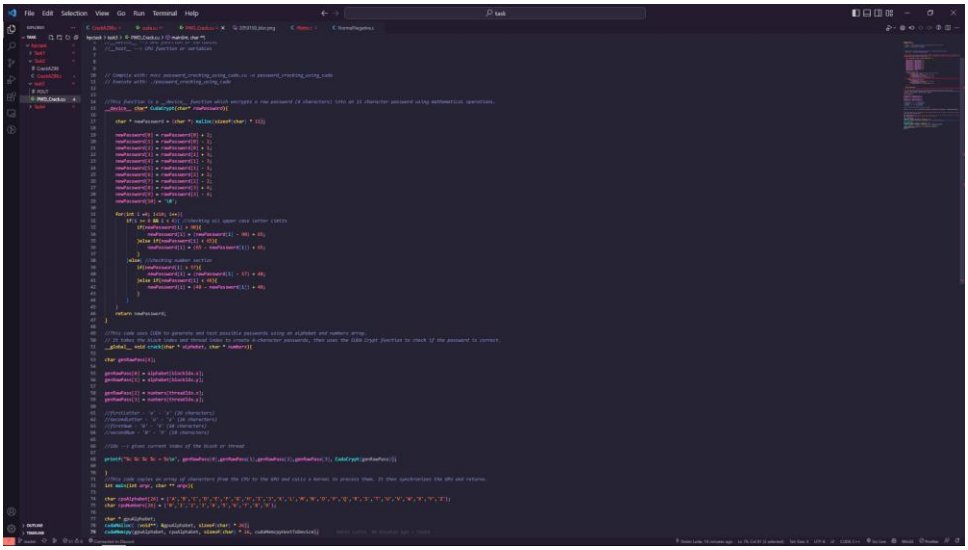
The crack function will loop through characters from a given start limit to end limit, and then loop through letters A to Z and numbers 0 to 99 and check if the given encrypted string is the same as the one given. If it is, it prints the plaintext, encrypted string, and the count. If the required password is found, all other threads will be cancelled. The prepareSliceList function is used to prepare the slice list, which is used to divide the workload to the thread. It assigns a range of characters to each of multiple threads, based on the number of threads. The first thread is assigned the range of characters starting with 'A' (ASCII value 65) and the subsequent threads are assigned the range starting with the ending character of the previous thread, plus one. The ranges are printed to the console.The main function runs the prepareSliceList function and prints the total number of solutions explored.

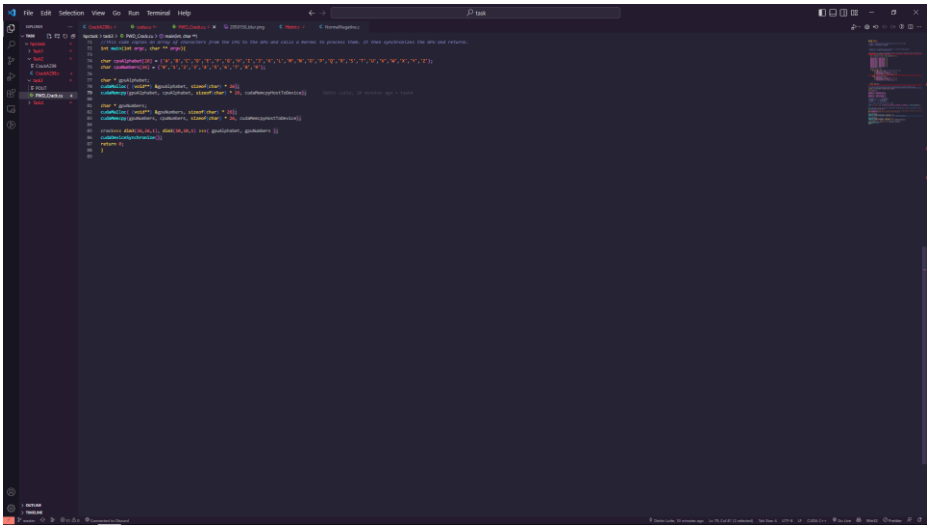
Task 3

3. Password Cracking using CUDA (35% - 100 marks)

Using a similar concept as question 2, you will now crack passwords using CUDA. As a kernel function

cannot use the crypt library, you will be given an encryption function instead which will generate a password for you. Your program will take in an encrypted password and decrypt it using many threads on the GPU. CUDA allows multidimensional thread configurations so your kernel function (which runs on the GPU) will need to be modified according to how you call your function.





Explanation in brief:

In this code, two arrays of characters are created on the CPU, one with the letters of the alphabet and one with the numbers 0-9. The CPU then copies these two arrays to the GPU using cudaMemcpy. The crack kernel is then launched with a grid of 26x26x1 blocks and a block of 10x10x1 threads. The kernel takes the blockIdx and threadIdx to generate a 4-character password, which is then encrypted into an 11-character password using the __device__ CudaCrypt function. The encrypted password is printed out and the kernel is synchronized. Finally, the GPU memory is freed and the program exits.

Task 4

4. Box Blur using CUDA (35% - 100 marks)

Your program will decode a PNG file into an array and apply the box blur filter. Blurring an imagereduces noise by taking the average RGB values around a specific pixel and setting it's RGB to the mean values you've just calculated. This smoothens the colour across a matrix of pixels. For this assessment, you will use a 3x3 matrix. For example, if you have a 5x5 image such as the following (be

aware that the coordinate values will depend on how you format your 2D array):

0,4 1,4 2,4 3,4 4,4

0,3 1,3 2,3 3,3 4,3

0,2 1,2 2,2 3,2 4,2

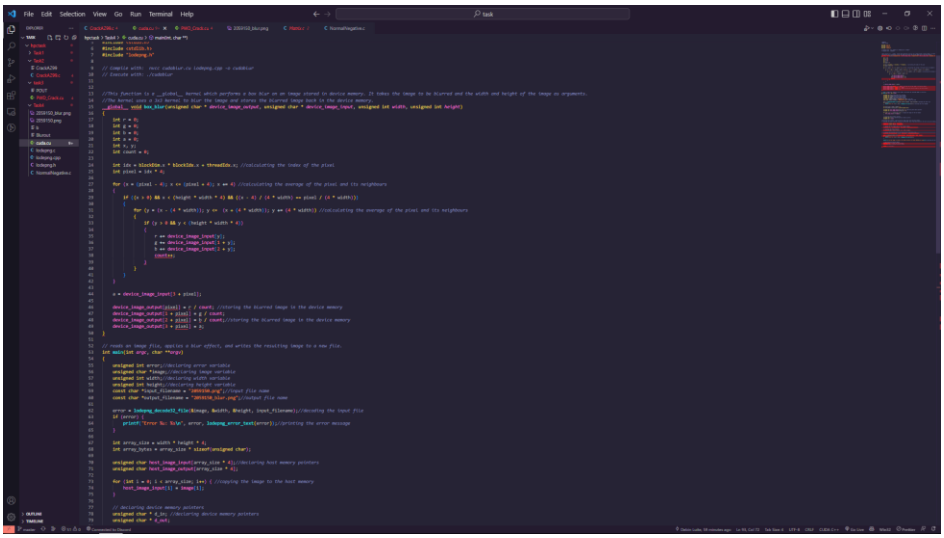
0,1 1,1 2,1 3,1 4,1

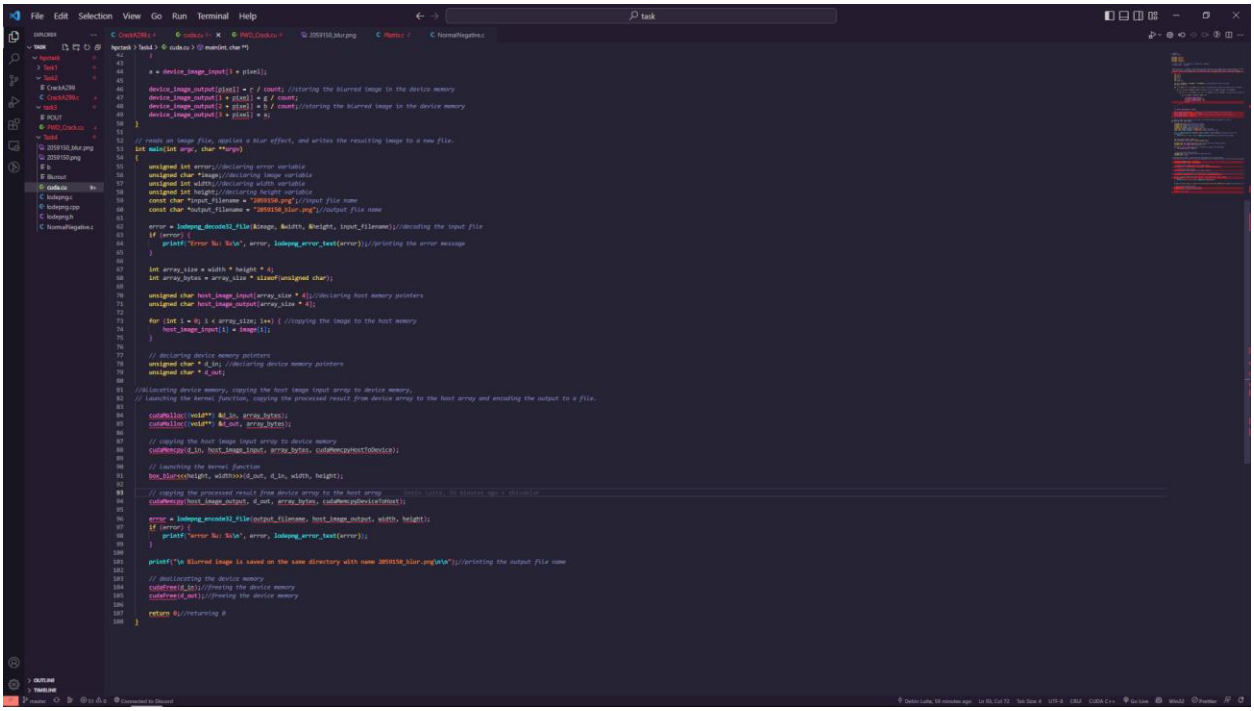
0,0 1,0 2,0 3,0 4,0

The shaded region above represents the pixel we want to blur, in this case, we are focusing on pixel 1,2 (x,y) (Centre of the matrix). to apply the blur for this pixel, you would sum all the Red values fromthe surrounding coordinates including 1,2 (total of 9 R values) and find the average (divide by 9). Thisis now the new Red value for coordinate 1,2. You must then repeat this for Green and Blue values.

This must be repeated throughout the image. If you are working on a pixel which is not fully surrounded by pixels (8 pixels), you must take the average of however many neighbouring pixelsthere are.Your task is to use CUDA to blur an image.

NOTE – this program should work with any amount of threads.





Explanation in brief:

The code above is used to blur an image stored in a file named 2059150.png. The code first reads the image file and stores it in a variable named image. The width and height of the image is then stored in the variables width and height.

It then creates two arrays, one to store the initial image in the host memory and one to store the processed image in the host memory.

The code then allocates device memory for the same two arrays and copies the initial image from the host memory to the device memory.

It then launches the box_blur kernel which performs a box blur on the image stored in the device memory. The blurred image is then stored back in the device memory.

Finally, the code copies the blurred image from the device memory to the host memory and then encodes the image and stores it in a new file named 2059150_blur.png. The code then prints the output file name and deallocates the device memory.

Input image

2059150_debin_luitel

Output Image