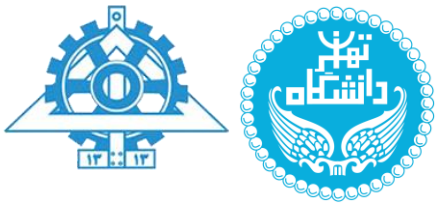
In the name of God

Computer Assignment 3



Faculty of Electrical and Computer Engineering
Operating Systems – Spring 2024

Assignment Coordinators:
Behrad Elmi - Pouria Tajmehrabi

Deadline: 23:59 on May 18th

Introduction

The goal of this exercise is to familiarize you with the basic concepts of multi-threaded design by solving a problem. In this exercise, you will apply filters to images. These images are in 24-bit bitmap format, and the code for reading these images is provided to you. You need to implement the application of filters on these images in both serial and parallel modes.

Exercise Description

In this exercise, you will work with applying filters to images, and after completing the steps, you will obtain a modified image as the result.

In the first part of your program, you will read input images and store the pixel values of its three-color channels in memory. In the second part, your program will operate on the image in grayscale to perform edge detection operations.

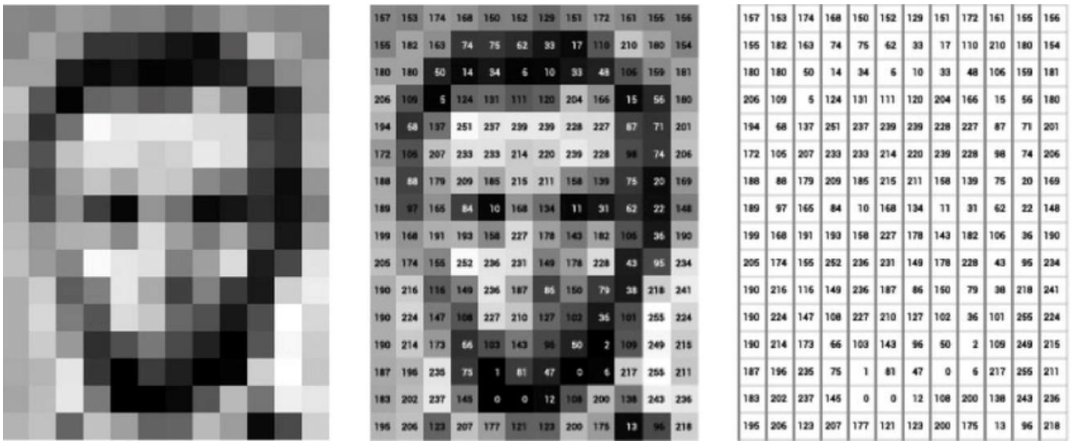
In each part of the program, you will apply relevant filters step by step. For this exercise, you are required to implement both serial and parallel methods and measure the speedup of the parallel version compared to the serial version.

Reading and Processing Images Using OpenCV

To read images, you will use the OpenCV library. This library helps you avoid the intricacies of implementing image file reading and saving, allowing you to focus more on performing desired operations on your images. You can install OpenCV using this link.

Please note that implementing filters using OpenCV's built-in functions is not permitted for this exercise.

You are required to implement the filters yourself.



Input Image

The image below is provided to you:

Part 1: Applying Filters on Color Image

In this part, you will apply the mentioned filters on the color image, such that the output of each filter will be the input for the next filter.

Horizontal Mirror Filter

This filter horizontally mirrors the image. Use the following formula to apply this filter:

$$outputImage(x, y) = inputImage(-x, y)$$

Vertical Mirror Filter

This filter vertically mirrors the image. Use the following equation to apply this filter:

$$outputImage(x, y) = inputImage(x, -y)$$

Applying Sepia Filter

This filter gives images a warm, vintage feel. By increasing the levels of red and yellow colors while decreasing the levels of blue and green, it imparts a reddish-brown tint reminiscent of old photographs. The filter works by applying a matrix of coefficients that determine how much of each color channel (red, green, blue) contributes to the final result.
To create a sepia filter, you can use the following formula for each pixel of an image:

$$TR = 0.393R + 0.769G + 0.189B$$
$$TG = 0.349R + 0.686G + 0.168B$$
$$TB = 0.272R + 0.534G + 0.131B$$

Part 2: Edge Detection on Grayscale Image

One of the applications of edge detection is in the automotive industry for self-driving cars. Edge detection helps vehicles gain a better understanding of their surroundings (such as lane markings, traffic signs, pedestrians, obstacles, etc.)**.**

In this part, you will implement one of the methods used for edge detection using the operations mentioned. Your input will be a grayscale image. To obtain a grayscale image, you can create a deep copy of the original image and convert it to grayscale using functions provided by OpenCV, or directly read the image in grayscale mode.

Box Blur Filter

Blur, also known as smoothing, is a step often performed on images before edge detection. This process enhances image coherence and reduces noise, thereby improving the edge detection process.

This filter computes the average of neighboring pixels for each pixel. In this section, you will implement a 3x3 Box Blur filter. This means each pixel will participate in averaging with its surrounding 3x3 neighborhood.

Simple averaging is sufficient, and there's no need to apply optimization algorithms for averaging matrices.

Consider the following example where the operation is performed on the central element of the matrix:

$$\begin{bmatrix} 0 & 2 & 3 \\ 5 & 1 & 6 \\ 7 & 6 & 0 \end{bmatrix} \xrightarrow{\text{Box Blur}} \begin{bmatrix} 0 & 2 & 3 \\ 5 & 3 & 6 \\ 7 & 6 & 0 \end{bmatrix}$$

$$= \frac{1}{9}(0+2+3+5+1+6+7+6+0) = 3$$

Converting Grayscale to Black and White

After blurring the image, the next step is to obtain a black and white image. For this purpose, thresholding is used. In this part, binary thresholding with a threshold of 127 will be applied, using the formula provided below:

$$dst(x,y) = \begin{cases} 255 & : src(x,y) > 127 \\ 0 & : otherwise \end{cases}$$

Output of this Stage:

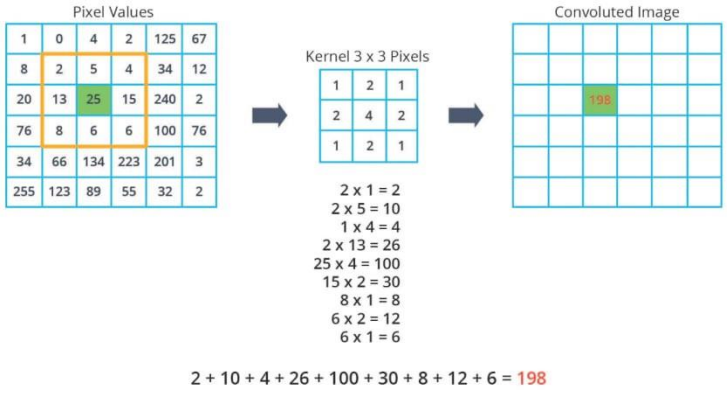
Laplacian Edge Detection

In this section, you will implement a filter that detects edges of objects. For this purpose, you will use the following kernel and perform a Convolution operation on each pixel of the image that meets the criteria. Note that Convolution requires values from the 8 neighboring pixels in addition to the pixel itself.

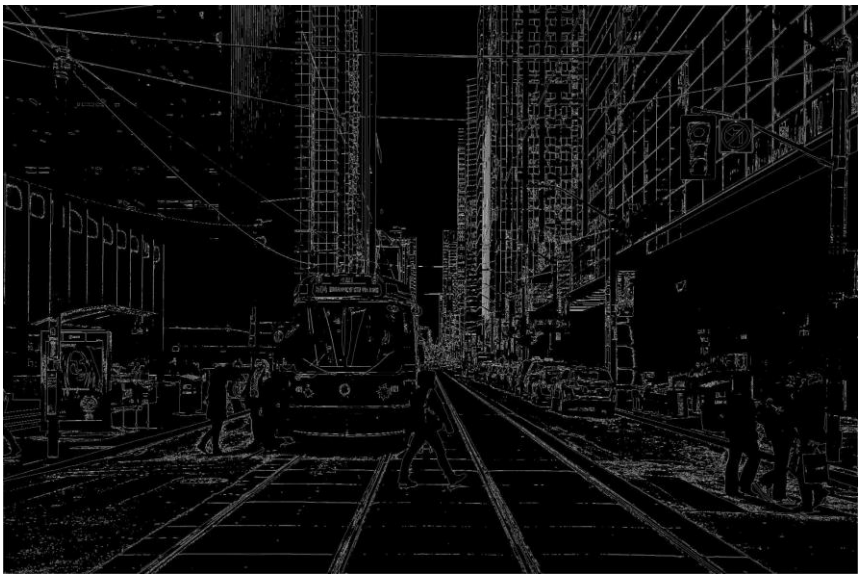
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Kernel for Laplacian Edge Detection

To perform Convolution, you can refer to the image below:



After blurring, converting to black and white, and finally applying the Laplacian filter, the output image will appear as follows:



Serial Implementation

In this section of the exercise, you will focus on implementing the program as a serial implementation. Strive to achieve the best execution time in this part of the exercise because the serial implementation should be optimized for comparison with the parallel versions. After this stage, identify the operations that consume the most execution time (hotspots) and explain them verbally during the presentation.

Parallel Implementation

In this section of the exercise, you will parallelize the operations performed in the functions identified as hotspots in the previous section. You need to determine the best combination of threads, data partitioning methods, and thread synchronization mechanisms and justify your choices. Finally, calculate the speedup of the parallel implementation compared to the serial implementation using the formula provided below, and report it in the specified format in your delivery:

$$speedup = \frac{serial\ execution\ time}{parallel\ execution\ time}$$

Notes

- Ensure that the output of your multi-threaded program matches your serial program.
- Note that this section of the exercise must be implemented as multi-threaded, and unacceptable sizes of implementations are not allowed.
- Pay attention that for parallelization of the project, you are only allowed to use the *PThread* library, and the use of other libraries for parallelization is not permitted.
- Your executable file name must be *ImageFilters.out* in both serial and parallel modes.

Input and Output of the Program

Finally, you must save the final output of each part of the project alongside your project. You can use OpenCV for this purpose. For the serial version, use the names *Serial_First.bmp* and *Serial_Second.bmp*, and for the parallel section, use *Parallel_First.bmp* and *Parallel_Second.bmp*. Your program should receive the input image name from the command line. Below is an example execution assuming the input image is named *ut.bmp* placed alongside your executable file. The specified output should be printed after executing each, for both serial and parallel implementations.

Sample Execution
./ImageFilters.out ut.bmp
Output Format
Execution Time : <execution_time>

Sample Output
Execution Time : 2.12

Additional Notes:

- Print all textual outputs of the program in the standard output stream.
- It is guaranteed that the inputs provided to your program are correct, so there is no need for input validation by you.
- Proper design, program efficiency, and breaking down the program into appropriate sections significantly impact grading.

Delivery Method:

- Ensure that your uploaded zip file is named OS_CA3_<SID>.zip and contains two separate directories. One directory should contain the serial implementation, and the other should contain the parallel implementation. Make sure the zip file does not include an outer folder layer, so that after unzipping, you directly obtain two folders: one for serial implementation and one for parallel implementation. Do not include input and output images in your uploaded file.

Acceptable Upload Structure:

```
OS_CA3_81019xxxx.zip
├── parallel
│   ├── main.cpp
│   └── makefile
└── serial
    ├── main.cpp
    └── makefile
```

Your program must run on the Linux operating system, compiled with g++ using the C++11 standard, and execute within a reasonable time for test inputs.

Your project must include a Makefile. Specify in the Makefile that you are using the C++11 standard. The executable file generated by your Makefile must be named *ImageFilters.out*.

Pay attention to instructions given during exercise justification sessions and in relevant forums as they are part of the exercise.

Translated by Ali Banihashemi