Parallel Digital Image Processing and Analysis

Introduction

The goal of this work is to implement image processing tools using sequential, multi-threaded and threadpool-base approaches, to study how to maximize eventual performance gains in scenarios where parallel and concurrent programming can leverage the potential of the hardware, and explain how and why each of these approaches impact in the performance of the solution.

Image filters

There are several image filters available in photo editing software. These filters transform an input image in an output image where at least some pixels are converted by the application of some calculation. Examples are, converting pictures from colour to grayscale, increase brightness or apply some type of distortion. For this project we will use the filters described in the following sections and note that we will consider images as matrices of pixels (Color objects) and an API with simple image processing features will be provided.

Brigthness

The values of the red, green and blue of each pixel are in the interval of 0 to 255. The increase in brightness is achieved by increasing this value. So, for the input figure 1, increasing the value of the pixels by 64 results in figure 2.

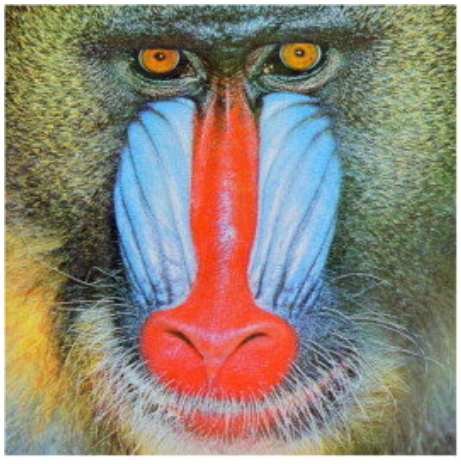

Figure 1: Original photo

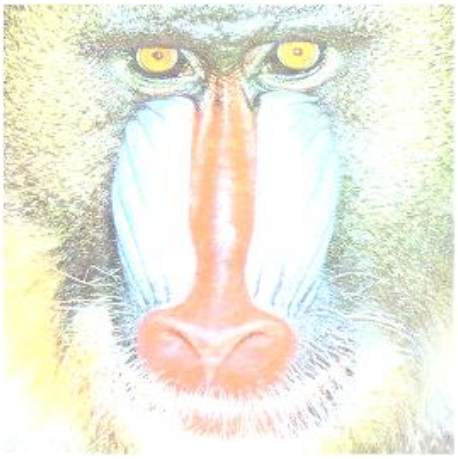

Figure 2: Brighter filter

Grayscale

Conversion to grayscale can be done by different methods. The one suggested here is to compute the average of the red, green and blue of each pixel and replace each of them by that value. More formally, for each pixel with red (r), green (g) and blue (b) values, compute:

$$\mathcal{A} = \frac{(r + g + b)}{3}$$

Replace each of the r, g and b by $\mathcal{A}$. So, for the input figure 1, applying this method results in figure 3.

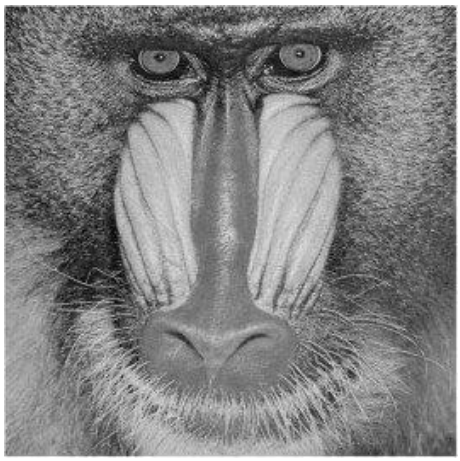

Figure 3: Grayscale filter

Swirl

The swirl filter or warp filter consists in rotating the image as presented in figure 4. For that purpose, one can use the formula explained next. Given the center of image (x_0, y_0) and given any point (x, y) in the image, $d = \sqrt{(x - x_0)^2 + (y - y_0)^2}$ and $\theta = \frac{\pi}{256} * d$ replace the values of x and y by the values of x' and y' such that:

$$x' = (x - x_0)\cos\theta - (y - y_0)\sin\theta + x_0$$
$$y' = (x - x_0)\sin\theta + (y - y_0)\cos\theta + y_0$$

So, for the input figure 1, applying this method results in figure 4.

Figure 4: Swirl filter

Glass

The glass filter swaps a pixel to a random close neighbor. The result is that the image seems to be displayed through a glass. More formally, we can achieve this by replacing every pixel (x, y) with a random near one with a maximum distance of, for example, 5 pixels. So, for the input figure 1, applying this method results in figure 5.

Figure 5: Glass filter

Blur

Here, the idea is to implement a filter to blur an image. Blurring an image can be done by averaging pixels with values of their neighbors. So if, for example we have pixel

$(x, y) = (113, 91, 94)$ where 113 is the value for the red, 91 the value for the green and 94 the value for the blue, and its neighbors are as described in table 1.

(142,113,115)	(150,120,120)	(157,127,127)
(113,91,94)	**(113,91,94)**	(123,101,104)
(129,108,113)	(128,107,112)	(133,112,117)

Table 1: 3x3 matrix of pixel contents with center in (113,91,94)

Now, the values for the red, green and blue are the following:

$$r = \frac{142 + 150 + 157 + 113 + \mathbf{113} + 123 + 129 + 128 + 133}{9} = 132$$

$$g = \frac{113 + 120 + 127 + 91 + \mathbf{91} + 101 + 108 + 107 + 112}{9} = 107$$

$$b = \frac{115 + 120 + 127 + 94 + \mathbf{94} + 104 + 113 + 112 + 117}{9} = 110$$

As a result of the operation, the pixel with $(x, y) = (113, 91, 94)$, should be replaced by $(x, y) = (132, 107, 110)$. The bigger the matrix of pixels being used is the more blurred the image becomes. It should be possible to choose a value m and to create the corresponding effect or each pixel using an $m \times m$ submatrix. For the edges, where a complete matrix is not available one should use only the available values. For the input figure 1, applying this method results in figure 6.

Figure 6: Blur filter

Conditional Blur

Conditional blur consists in applying Blur only when some condition is satisfied. For example, if a pixel has some amount of red it will be blurred, otherwise we skip that pixel. If we define the red threshold as, for example 200, a pixel $(x, y) = (r, g, b)$ where $r > 200$ will have the filter applied, otherwise it will be ignored. One example where only red pixels are blurred is given in figure 7.

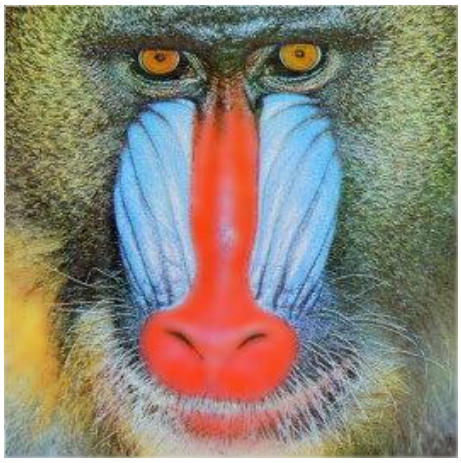

Figure 7: Conditional Blur filter

Sequential implementation

The first task is to create a sequential implementation of the filters. A simple API is provided to translate between images to matrices of Color objects and vice versa along with reading and writing images from and to the filesystem. There are also references that can help to understand such implementations for example the book Digital Image Processing 4th edition [1] and the book Computer Science: An Interdisciplinary Approach [2] and the web site of the former book [1].

Multithreaded implementation

The second part of this project is to create a multithreaded implementation of the filters were the images are divided and split by different threads that should optimize the application of the filters relying on multicore architectures. Defining the number of threads and the amount of work for each one of them is part of the projects' objectives, thus the data decomposition strategy and the data structures involved should be carefully choose, understood and analyzed to obtain the best possible performance for each scenario.

Threadpool-based implementation

The third part of this project is to use threadpools to solve the problem. Meaning that thread creation and teardown should be of the responsibility of a threadpool. The amount of work to be fed to the threadpool should be carefully analyzed. In this part of the project different uses of threadpool should address and analyzed namely, Executor-based, ForkJoinPool-based and at the highest level, using CompletableFutures. Note that the use of strategies such as work-stealing are carried only in some types of threadpools and thus, depending on the applications scenario, different threadpools may have different performance results and it is a prime goal to understand these differences.

[1] `https://introcs.cs.princeton.edu/java/31datatype/`

Garbage Collector Tuning

Java provides different garbage collectors which run simultaneously with our applications that clean memory by using different strategies. Some divide the memory area in parts, use one or more threads, etc. It is possible to choose and configure garbage collectors in Java, evidences of such work should be provided and the choice of an appropriate garbage collector must be justified.

Generation of results

The last part of this project consists in measuring and analyzing the results of the different approaches. The developed solution should measure time of execution of the application of the filters for several images with different sizes in each of the different approaches. All the data should be gathered in automatically generated tables and the results should be used in the final report when explaining results.

Starter Code

Starter code is provided. It includes both a simple API to the images and an example of the application of one of the filters. The API is briefly described next:

Color[][] loadImage(String filename) : given the path to an image file in **filename** returns an array **Color[][]** of pixels of that image.

void writeImage(Color[][] image,String filename) : given an array of **Color** objects and a path to a file in **filename**, writes the array to that **filename** creating an image in the filesystem.

Color[][] copyImage(Color[][] image) : creates and returns a copy of an array of **Color** objects. This is useful when it is necessary to work over a copy of the image.

The file **ApplyFilters.java** uses the simple API described before to create an object **Filter** that reads an image from the filesystem, applies the *Brighter Filter* to that image and writes the result back to the filesystem.

Grading

Project is to be solved individually or in groups of maximum, four (4) elements.

Description	Percentage (%)
Implementation of image filters	15
Multithread-based processing	20
Threadpool-based processing	30
Analysis of results and discussion (includes final report)	35

Scheduling

The final deadline for the submission of this project is 28^{th} of April of 2024 and the presentations will take place on the following days. There will be an optional and partial presentation on the week of the 8^{th} of April for feedback on the progress of the project.

Code of honor

In "Código de boas Práticas de Conduta" from 27th of October of 2020, it is stated that students must submit a declaration as described in the Article 8 for each submitted project. This declaration is a signed commitment of the originality of the submitted work. Failure to submit this declaration will remove the work from evaluation. Submitting a work that does not comply with the signed declaration will have legal consequences.

References

[1] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson, 2018.

[2] Robert Sedgewick and Kevin Wayne. *Computer Science: An Interdisciplinary Approach*. Addison-Wesley Professional, 1st edition, 2016.