PC-2019/20 Course Project Template

Leonardo Casini
E-mail address
leonardo.casini1@stud.unifi.it

Harjinder Singh Sandhu
E-mail address
harjinder.sandhu@stud.unifi.it

Abstract

We present an image Kernel Precess Sharpening algorithm for reducing execution time, exploiting details of the code and analyzing different results. Our system makes use of parallelisms in computation via the JAVA thread programming model. This implementation treats portions of the tar-get image as independent input sub-matrix for the sharpening algorithm and finally merge partial results to accomplish the task. We compare the performance of the parallel approach running on the CPU with the sequential implementation across a range of image sizes

1. Introduction

Before getting into the act of sharpening an image, we need to consider what sharpness actually is. The biggest problem is that, in large part, sharpness is subjective. Sharpness is a combination of two factors:

- **Resolution** is straightforward and not subjective. It's just the size, in pixels, of the image file. All other factors equal, the higher the resolution of the image the more pixels it has the sharper it can be.
- **Acutance** is a subjective measure of the contrast at an edge.There's no unit for acutance you either think an edge has contrast or think it doesn't. Edges that have more contrast appear to have a more defined edge to the human visual system.

Sharpening then, is a technique for increasing the apparent sharpness of an image an in Image processing is accomplished by doing a convolution between an image and a kernel(small matrix).

1.1. Convolution

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel.The general expression of a convolution is

$$g(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) \star f(x+dx.y+dy) \tag{1}$$

where g(x,y) is the filtered image, f(x,y) is the original image, w is the filter kernel.
Depending on the element values, a kernel can cause a wide range of effects: for sharpening we consider the kernel in 2.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2}$$

2. Implementation

As described above, to apply sharpening , we first convert the image color space from RGB to HSB, then the algorithm can be applied to the brightness or value channel without resulting in changes to the hue and saturation of the image.
Both sequential and parallel implementation are tested and timed over the equalization of the B channel.

2.1. Sequential Version

For an image of size wŒh, we form the output image using the following algorithm,

- **B** : input Brightness channel of dimension w Œ h. Express in percent value.
- **I**: B converted in greyscale.
- **K**: kernel matrix of dimension 3x3.
- **B'**: output Brightness channel of dimension w Œ h.

Algorithm 1 Serial Sharpen Algoritm

```
1:  procedure SHARPEN(B)
2:      for i = 0 → B.width do
3:          for j = 0 → B.height do
4:              I[i, j] ← round(B[i, j] ∗ 255)
5:          end for
6:      end for
7:      for i = 0 → B.width do
8:          for j = 0 → B.height do
9:              sum = 0
10:             for l = 0 → K.width do
11:                 for m = 0 → K.height do
12:                     sum ← sum + I[x, y] ∗ K[l, m]
13:                 end for
14:             end for
15:             B'[i, j] ← round(sum/255)
16:         end for
17:     end for
18:     return B'
19: end procedure
```

The procedure calculate the output value pixel in base of the neighbours as it show in fig. 1.

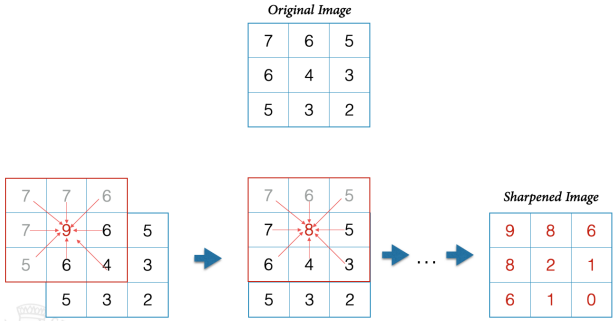

Figure 1. An example of procedure

2.2. Parallel Version

We developed two different classes of threads:

- **PartialMatrixBuilder** : given a portion of the input, calculates the partial matrix in greyscale.
- **ConvoluteThread**: given a portion of the input in greyscale, calculates the convolution and return the filtered partial matrix in Brightness channel value.

Each thread has its own portion of the image with $imageWidth/numberThreads$ as *width*. So, we define for each thead:

- **start** : input index, denotes the portion of B assigned to the specific thread.
- **end**: input index, denotes the portion of B assigned to the specific thread.

In algorithm 2 is shown the run method of the first type of thread.

Algorithm 2 Run of PartialMatrixBuilder

```
1: procedure RUN(B, start, end, I)
2:     for i = start → end do
3:         for j = 0 → B.height do
4:             I[i, j] ← round(B[i, j] ∗ 255)
5:         end for
6:     end for
7: end procedure
```

Next is shown the run method of second thread:

Algorithm 3 Run of ConvoluteThread

```
1:  procedure RUN(I, K, start, end, B')
2:      for i = start → end do
3:          for j = 0 → I.height do
4:              sum = 0
5:              for l = 0 → K.width do
6:                  for m = 0 → K.height do
7:                      sum ← sum + I[x, y] ∗ K[l, m]
8:                  end for
9:              end for
10:             B'[i, j] ← round(sum/255)
11:         end for
12:     end for
13: end procedure
```

In fig 3 it is shown the scheme of the parallel implementation.

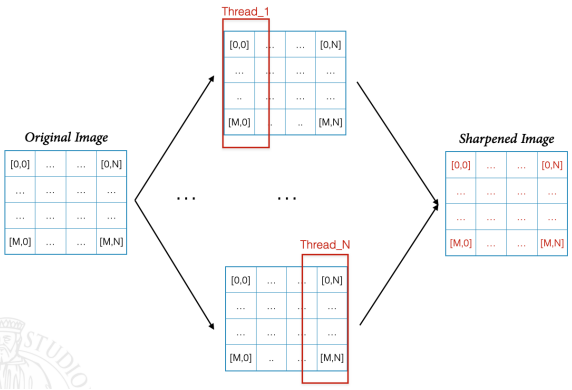

Figure 2. Parallel implementation

3. Results

All the results, in terms of optimal number of threads and execution time, are obtained on a machine with Intel(R) Core(TM) i7 @ 2.80GHz CPU.

3.1. Number of thread

Using an image of size 10000x10000 we tested the execution time with different number of threads from 1 to 1000. It's known that increasing the number of threads increases performance, but after the value of 8 threads we have no particular increases. So we decided to use this number of threads in the next tests.

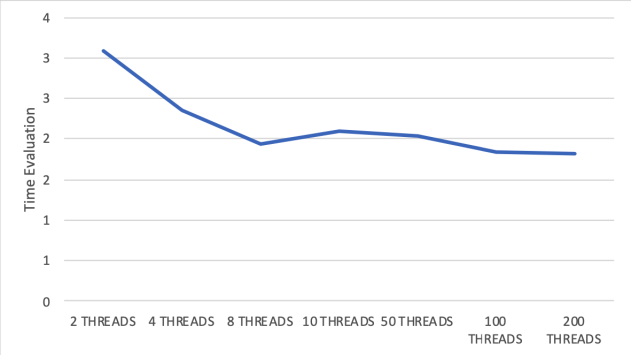

Figure 3. Comparison between different number of threads

3.2. Serial vs Parallel

Varying image size, we discover that with image larger than 1728x1728 pixels the parallel algorithm is always faster of the serial. To quantify this improvement we computed the speed up factor between the two implementation as: Parallel execution time/Serial execution time. The result is observable in fig. 5.

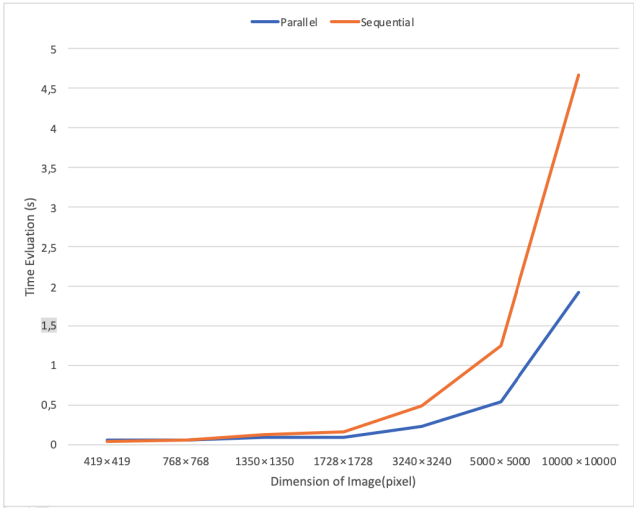

Figure 4. Comparison between parallel and sequential version

Figure 5. Speed up factor of Parallel implementation over serial

References

[1] Kernel (image processing), https://en.wikipedia.org/wiki/Kernel_(im

[2] Materiale didattico - Parallel Computing