Kernel Image Proccesing

Elmir Šut

UP FAMNIT
E-mail: *elmirsutisz@gmail.com*

23.12.2019

This report will present a framework for calculating convolution of each pixel in an image using convolution matrix(kernel). In particular, the main focus was on using edge detection convolution matrices when solving calculations sequentially, in parallel using multiple threads and with message passing between multiple single-threaded processes (MPI). Measurements were recorded and played key role in deciding our final solution.

Using simple GUI (graphical user interface) results will be shown interactively with a final image and time taken to process it using the aforementioned programs.

1 Introduction

The smallest segment of an image is a pixel, depicted as square. Every single image is two-dimensional graphics determined by width and height, measured in pixels. Each pixel has strictly determined place and it is defined by number of bits used to save it in memory.

Pixels in monochrome images consist of one value which represents light intensity or specific shade of grey, but in color images three values (Red, Green and Blue) are needed to create one pixel. Therefore color image is consisted of three same-size matrices, each representing one color among aforementioned color channels.

Convolution is a process where we add value of each pixel weighted by convolution matrix to its local neighbours where as a result we get new value for current pixel.

The general expression of a convolution is:

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

where *g(x,y)* is the convoluted image, *f(x,y)* is the image to be convoluted, ω is the convolution matrix. Every element of the convolution matrix is considered by $-a \leq s \leq a - a \leq s \leq a$ and $-b \leq t \leq b - b \leq t \leq b$. [1]

[1] WIKIPEDIA: https://bit.ly/2EKU1dh

Algorithm 1 Sequential Convolution

```
1:  function CONVOLUTE(INPUT, WIDTH, HEIGHT, KERNEL, KERNELW, KERNELH)
2:      smallWidth ← width - kernelWidth + 1
3:      smallHeight ← height - kernelHeight+ 1
4:      output ← heightXwidth
5:      for i ← 0 to smallWidth do
6:
7:          for j ← 0 to smallHeight do
8:              output(i + 1, j + 1) ← SPC(input, i, j, kernel, kernelWidth, kernelHeight)
9:          end for
10:     end for
11:
12:     return output
13:
14:     function SPC(INPUT, X, Y, KERNEL, KERNELW, KERNELH)
15:         accumulator ← 0
16:         for i ← 0 to kernelWi do
17:
18:             for j ← 0 to kernelH do
19:                 accumulator ← accumulator + $input(x+i, y+j) * kernel(i,j)$
20:             end for
21:         end for
22:         return accumulator
```

▷ Total complexity of this algorithm will be defined by smallWidth, smallHeight, kernelW and kernelH so we can represent it as:
$O(smallWidth * smallHeight * kernelW * kernelH)$

▷ Considering special case when image is square-shaped(*width* == *height*),, and kernel is square-shaped(*kernelW* == *kernelH*), then our complexity would be $O((N * M)^2)$