Kernel Image Processing

Luka Uršič
89221145
UP Famnit
E-mail: 89221145@student.upr.si

May 19, 2024

Abstract

In this paper, I present how to use kernel image processing to modify an image. I explain how kernel image processing works, how I implemented it, and the time results I obtained from running it sequentially, in parallel, and with distributed computing. I compare the results and conclude which method is the best for this specific task.

1 Introduction

An image kernel is a small matrix used to apply effects like the ones you might find in popular photo manipulation software, such as blurring, sharpening, outlining, or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context, the process is referred to more generally as "convolution".

Setosa. *Image Kernels Explained Visually.* Accessed: 2024-5-16. 2015. URL: `https://setosa.io/ev/image-kernels/`

2 Implementation

I implemented Kernel Image Processing in Java and used the Swing and AWT libraries to display the images. I ran the program on a single machine and on a cluster of machines to compare the results. I measured the time it took to process the image and compared the results.

I created a class called ImageProcessor that contains the methods applyKernelToPixel, applyKernel, applyKernelSequential, and applyKernelParallel. The applyKernelToPixel method applies the kernel to a single pixel, the applyKernel method applies the kernel to the entire image, the applyKernelSequential method applies the kernel to the image sequentially, and the applyKernelParallel method applies the kernel to the image in parallel.

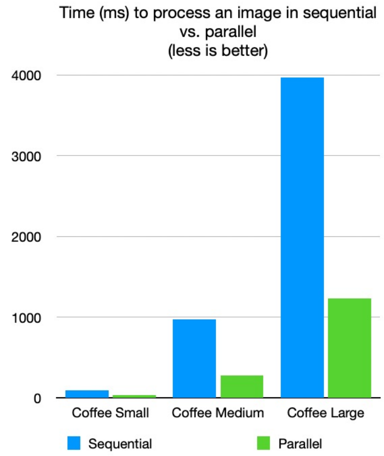
Algorithm 1 Pseudocode for ImageProcessor.java

```
Class ImageProcessor
Function applyKernelToPixel(image, kernel, result, x, y)
for each value in the kernel do
    Multiply the corresponding pixel color by the kernel value
    Add the result to r, g, b
end for
Clamp r, g, b between 0 and 255
Set the pixel in the result image to the new color
End Function

Function applyKernel(image, kernel)
for each pixel in the image do
    Apply the kernel to the pixel
end for
End Function

Function applyKernelSequential(image, kernel)
Apply the kernel to the image
Return the result image
End Function

Function applyKernelParallel(image, kernel)
Use a ForkJoinPool to apply the kernel to each pixel in parallel
Return the result image
End Function
End Class
```

3 Results

I tested the program on my computer in sequential mode and in parallel mode with three picures of different sizes. Results in sequential mode were following the number of pixels in the image. The results in parallel mode were faster than in sequential mode. The difference was significant. It noted speedups of up to 4x. What surprised me, is that the speedup was present with small and large images.

	Sequential	Parallel	Pixels
Coffee Small	93	34	312320
Coffee Medium	972	282	4392000
Coffee Large	3966	1233	18226192



4 Conclusion

References

Setosa. *Image Kernels Explained Visually*. Accessed: 2024-5-16. 2015. URL: `https://setosa.io/ev/image-kernels/`.