Final Report Rough Draft

Marcelino Pozo  
Computer Science Student  
University of Central FloridaOrlando, United States  
ma986783@ucf.edu

Jared Reich  
College of Engineering and Computer Science  
University of Central FloridaOrlando, United States  
ja548470@ucf.edu  
  
line 1: 3rd Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID  
  
line 1: 4th Given Name Surname  
line 2: *dept. name of organization (of Affiliation)*  
line 3: *name of organization (of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

*Abstract*—This study investigates the impact of using parallel processing over linear processing for image convolution. In this research, we compared the performance of parallel processing with three threads against traditional linear processing for convolution tasks. Our results demonstrate a significant improvement in computational speed when utilizing parallel processing, with a threefold increase observed in processing speed compared to linear processing. This finding highlights the effectiveness of parallel processing techniques in accelerating image convolution tasks, offering promising implications for real-time image processing applications.

# Introduction

Image processing tasks, such as convolution, play a crucial role in various fields, including computer vision, medical imaging, and digital photography. With the ever-growing demand for faster and more efficient image processing algorithms, researchers have explored parallel processing techniques to accelerate computational tasks. In this context, this paper investigates the efficacy of parallel processing over traditional linear processing for image convolution tasks, focusing on the implementation in Python. Conventionally, image convolution involves applying a filter to each pixel of the image, which can be computationally intensive, especially for large images or complex filters. To address this challenge, parallel processing offers a time saving solution by distributing the workload across multiple threads simultaneously. In this research, we specifically explore the performance gains achieved by employing parallel processing in a Python-based image convolution framework versus linear processing in Python. Our approach utilizes three threads to apply a sharpening filter to the red (R), green (G), and blue (B) channels of an image simultaneously. Although the current implementation processes one image at a time, the utilization of parallel processing allows for faster computation by leveraging the computational resources of modern multi-core processors efficiently. By comparing the computational speed of parallel processing with three threads against traditional linear processing, we aim to quantify the performance improvement achievable through parallelization. The findings of this study not provide valuable insights for optimizing image processing applications.

# Problem Statement

The goal of this project is to demonstrate the difference between single and multi-threaded image convolution. Image convolution involves applying a matrix (in this case a 3x3 matrix of signed integers) known as a kernel to an image’s individual RGB pixel values. To achieve the desired effect convolution must be done on every pixel in the image which can be very computationally expensive when considering that an image is made up of three two dimensional arrays representing the color channels of the image.

[2] Image Convolution


1[2] Image Convolution

To showcase the difference in compute-time we built a single threaded version as a control. Both the single threaded and multi-threaded versions are able to handle applying a sharpening effect to colored images.

## Single Threaded Colored Convolution

The single threaded version of this project was built first and served as a baseline for the multithreaded CPU and later GPU versions. All versions were built in Python and utilized libraries such as NumPy and matplotlib to deal with computations.

This version presented us with mostly straightforward tasks and challenges. The main tasks of performing the convolution on a given image were as follows:

* Take an image from a given file destination and convert it to a 3D array.
* Split the 3D array into three 2D array’s representing the R, G and B channels of the image.
* Apply convolution using a sharpening filter matrix to each channel sequentially.
* Recombine the image channels and output the filtered image
* Record the time elapsed.

Each of these steps was built into a single source file called single.py and took only 98 lines of code. However, we did face some challenges when building out this version.

Since it was the first version of the project we had to do research on what library would allow us to accomplish convolution the cleanest. We eventually settled on the using the copy library to create deep copies of each of the channel arrays of the image. According to python documentation a deep copy “constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original”[1]. This was necessary as we wanted to create entirely new objects when copying over the data rather than just modifying the original 3 dimensional array, which would be bad practice. This does however increase the space in memory needed which will be discussed further down and is a problem with all versions of this project.

## Multi Threaded Colored Convolution

To test multithreading and incorporate it into this project the most simple baseline solution we came up with as a starting point was the most natural. Taking the R, G and B channel and running the convolution computations on them in parallel.

Using the joblib Python library for parallelization and the linear version of this project made this step a lot easier. It essentially has the same tasks as the linear version with some added complexity.

* Take an image from a given file destination and convert it to a 3D array.
* Split the 3D array into three 2D array’s representing the R, G and B channels of the image.
* Start three threads corresponding to the three channels and associate each 2D channel array with each thread.
* Apply convolution using a sharpening filter matrix to each channel in parallel
* Recombine the image channels and output the filtered image
* Record the time elapsed.

The biggest challenge with the multi-threaded version was Python's GIL means that even in multi-threaded applications, only one thread can execute Python bytecode at a time. For CPU-bound tasks such as image processing, this can limit the effectiveness of threading since threads might not actually run in parallel but instead take turns executing. The overhead of managing threads and context switching can make the multi-threaded version slower for such task To get around this we used joblib which made the threads run as expected since we had three threads it cut the time in thirds

TODO: Convolution of a colored image using multiple threads per channel

Tasks

Challenges

TODO: Convolution of a colored image using multiple threads per channel on the GPU

Tasks

Challenges

# Related Work

Before starting on this project lots of research was done in order to set the team up for success when building out our solution to this problem

## Research into Convolution

Convolution was not a topic all members of the group we’re familiar with so research into what it is was necessary. Image convolution was specifically what was looked into as it would be the goal of this project.

Image convolution is a fundamental operation in image processing and computer vision. It involves applying a kernel (also known as a filter or mask) to an input image to perform operations like blurring, sharpening, edge detection, or other enhancements.

## Research into Multithreading

Research into multithreading implementation in Python was also necessary for this project as we had not covered a Python implementation in class.

To achieve multithreading we used joblib which is particularly useful for parallelizing simple operations across multiple CPU cores efficiently. It provides a simple interface for parallelizing tasks and is commonly used for embarrassingly parallel tasks such as independent function calls or loops which made it perfect for convolution

## Research into Kernel Types

Finally using specific values for the kernel matrix was important in achieving the desired effect of sharpening so research into this area was a must.

# Technique

## Convolution Operation

The core of the proposed technique lies in convolution operation. A given set of predefined sharpening filters are applied to each individual color channels (Red, Green, Blue) of the image using a multithreaded implementation. Convolution involves element wise multiplication and summation to enhance details in the image.

## Multithreading

To take advantage of the parallel processing capabilities of modern computing, the image sharpening process is parallelized across the color channels. The implementation uses the *joblib* library to concurrently apply the convolution operation on the red, green, and blue channels. The parallelization of the technique improves the overall speed of image sharpening.

## Parallelizing Convolution Steps

Using imageio python package read image in. Convert the input into array. Using the copy package from python, create a deep copy of each color channel in the image. (The reason for doing the Deep copy of each splice instead of passing by reference is because of race conditions when each thread trying to access the same memory at the same time.) Three kernels are defined to pass to each thread. The function Parallel is then called from the package, joblib, and passed applyMultithread function with kernel, Deep copy images, and stride of 1. Sizes of Kernel, and Image are stored and a 2D arrays of zeros is initialized for storing the final image. Two nested for loops are used to iterate through the image which sums the multiplication of the kernel and image and store the resulting pixel value in the convolved image array. The results are returned to the doMultithread scope and stacks the RGB channels back together using numpy and returns the resulting image to main scope and records the end time.

# Evaluation

## Experimental Setup

The convolution is done to a sample image called (blurryFox.jpg). The image is 900 x 601 pixels. A file named “single.py” ran the entire convolution on a single thread in the main function as a control to determine runtime analysis. The parallelized convolution technique was done in “multi.py”

## Runtime Evaluation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time in (seconds) | Trial 1 | Trial 2 | Trial 3 | Average |
| Single Thread | 8.779 | 7.960 | 7.853 | 8.197 |
| Multithread | 3.434 | 3.322 | 3.323 | 3.360 |

The calculated speed up is a 41% reduction in time versus a procedural single threaded implementation of convolution of a colored image.

# Discussion

## Perforamnce

The parallelization of the convolution operation shows performance gains. Notably the performance gain is much more pronounced with larger images. Even with a image that is

900 x 601 pixels shows 41% increased speed across the board.

A possible improvement can be made to the current implementation of the multithreaded program. That improvement would be parallelizing the double for-loop in the applyMultiThread. The issue that could arise is a trade-off of using more memory to avoid deadlocking and race conditions. Would the benefits of parallelizing the for-loop convolution step create better runtime? Probably not.

# Conclusion

This report presents a multi-threaded image sharpening technique utilizing convolution filters. The results show the effectiveness of the approach in image sharpness. The convolution operation and parallelization of it, allows for the technique to be suitable in real time applications. Future iterations can improve upon small performance gains.

##### References

1. “Copy - Shallow and Deep Copy Operations.” *Python Documentation*, docs.python.org/3/library/copy.html. Accessed 8 Mar. 2024.
2. Sameer. “Image Convolution from Scratch.” *Medium*, Analytics Vidhya, 5 Jan. 2021, medium.com/analytics-vidhya/image-convolution-from-scratch-d99bf639c32a.