Enhancer for Scanned Images

Algorithm Engineering 2023

Sheela Orgler
Friedrich-Schiller-University Jena
Germany
sheela.orgler@uni-jena.de

ABSTRACT

This paper presents an implementation to improve the quality of scanned images using image processing. There are several methods to improve the quality of images like image enhancement, noise reduction, and image segmentation.
This implementation uses a median filter for noise reduction and adaptive mean thresholding to improve text readability. Furthermore, the focus is on improving performance by making use of parallelism, vectorization, and focusing on writing cache-friendly code.
The results of the performance optimization are shown in detail explaining the used methods like OpenMP and comparing execution time with and without the use of optimization techniques.

KEYWORDS

enhancer, scanned images, image processing, median filter, adaptive thresholding, C++, CMake, OpenMP, parallelism, vectorization, cache

1 INTRODUCTION

Over time image processing has become more and more important. Analyzing data from images is relevant in everyday applications across a broad range of fields from medical analysis to face recognition. Image processing describes the use of a computer to process images with the use of an algorithm. This paper focuses on image processing to improve the quality of scanned images. More precisely, it focuses on noise reduction and image enhancement.

1.1 Background

The improvement of scanned images revolves around the topic of image processing. This involves enhancing the quality of digital images obtained from scanned documents or photographs. One of the main challenges is the presence of various disruptive elements, such as noise, blur, distortion, and color cast. Several techniques have been developed, to address these issues.
This paper focuses on noise reduction and adaptive thresholding. Noise reduction is used to reduce noise from an image, for aesthetic as well as practical purposes. Noise in an image is a random variation of brightness or color information. This can reduce the quality of the image. Image denoising uses filters to remove random noise as far as possible. The implementation presented in this paper uses a median filter, which is a non-linear filter.
Adaptive thresholding is used for image segmentation. Image segmentation includes a set of techniques for image processing. With adaptive thresholding, a threshold is computed for each pixel. Each pixel is replaced with a black one if it's less than the computed threshold and with a white one if it's greater than the threshold. This results in a binary image with separate dark and bright regions. Adaptive thresholding is used to improve clarity. Especially, for images of documents it can be useful to improve the readability of the text.

1.2 Related Work

For this project, several existing projects and papers have been evaluated. Specifically, regarding adaptive thresholding, several methods can be used.
The paper Adaptive Thresholding: A comparative study from the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) [2] introduces and compares several different methods for adaptive thresholding. The implementation presented uses an adaptive mean thresholding technique, where the threshold is calculated for each pixel using the mean of the respective neighborhood. This method is introduced in an article about image binarization [3], which is the separation of black and white pixels in an image. Image filters for noise reduction can be divided into linear and non-linear filters. The article Implementation of PPM Image Processing and Median Filtering [1] presents an implementation for a median filter, which was also used in this implementation.

1.3 Contributions

This project uses two methods to improve the quality of scanned images. First, a median filter is used for noise reduction. Then, adaptive mean thresholding is applied to dynamically improve text readability. The implementation makes use of parallelism, vectorization, and cache-friendly code to improve performance and execution time.

2 THE ALGORITHM

The implementation provides a console application that takes an image as input. Image processing techniques are applied to the input and produce an improved output image.

2.1 Input and Output

The algorithm uses ppm images as input and output. PPM (Portable Pixel Map) is a file format used to store digital images. It is commonly used in computer graphics and image processing.
PPM images all have the same simple structure. They are uncompressed and consist of a plain text header followed by the binary pixel data. The header contains metadata about the image, such as the image width, height, and maximum pixel value. There are two variants of PPM: PPM P3 and PPM P6. PPM P3 is the ASCII version of the format, where the pixel values are represented as ASCII characters. PPM P6 is the binary version of the format, where

the pixel values are represented as binary data. The format stores the respective RGB values from 0 to 255 for each pixel.
When applying the adaptive mean thresholding method, the ppm image is converted to a PGM image, which is a grayscale image. For each pixel, the grayscale is stored.
To interact with the program a console application is provided. The image path is taken as input and the processed image is stored in the images folder. The program accepts images in the ppm format described previously.

2.2 Median Filter

A median filter is a method used for image processing to remove noise from an image. The filter works by sliding a window over the image and replacing the center pixel of the window with the median value of the neighboring pixels. The neighboring pixels are sorted and the middle value is taken to replace the central pixel. A sorting algorithm is used to reorganize the respective values. The median filter is explained in detail by the Algorithm 1.
Unlike linear filters that take the weighted average of the pixels, the median filter simply selects the middle value of the pixel values within the window. The median filter is effective in removing impulse noise, which is random noise that manifests as bright or dark pixels that are not present in the original image. It is useful in preserving sharp edges and fine details in an image, as it does not blur the edges.

Algorithm 1 Median Filter

```
function MedianFilter(image, width, height)
    result ← empty image of same size as image
    for y in height do
        for x in width do
            // Compute median of pixels in the
            // neighborhood of current pixel
            window ← empty list
            for dy ← −1 to 0 do
                for dy ← −1 to 0 do
                    if i and j are inside image then
                        add image(i, j) to window
                    end if
                end for
            end for
            sort window
            median ← value in the middle of window
            // Update result image with median value
            result(x, y) ← median
        end for
    end for
    return result
end function
```

2.3 Adaptive Mean Thresholding

Adaptive mean thresholding is used to improve the lighting conditions of the image. It is a technique used in image processing to separate the foreground from the background of an image. It is typically used for image variations in lighting and contrast across different regions.
The algorithm works by dividing the image into small regions and computing a threshold value for each region based on its mean pixel value. Pixels within a region that have a value above the threshold are considered part of the foreground, while pixels with a value below the threshold are considered part of the background. The implementation uses an adaptive mean threshold. The value for the threshold is calculated by subtracting the mean of the neighborhood area of each pixel from a constant.
To apply adaptive thresholding to the image the ppm image is converted to a PGM image. As the focus is on scanned images it is important to enhance the clarity of the text displayed in the image.

Algorithm 2 Adaptive Mean Thresholding

```
function AdaptiveMeanThresholding(image, blocksize, offset)
    for each pixel (x, y) in image do
        // Compute local threshold for current pixel
        sum ← 0
        count ← 0
        for i ← y − blocksize/2 to y + blocksize/2 do
            for j ← x − blocksize/2 to x + blocksize/2 do
                sum ← sum + image(i, j)
                count ← count + 1
            end for
        end for
        mean ← sum/count
        threshold ← mean − offset
        // Apply threshold to current pixel
        if image(x, y) < threshold then
            image(x, y) ← 0
        else
            image(x, y) ← 255
        end if
    end for
    return image
end function
```

2.4 Performance improvements

The implementation focuses on performance optimization. Performance improvement is the process of optimizing speed and efficiency. There are several approaches to optimizing an implementation, including algorithm optimization, parallelization, caching, and code optimization.

OpenMP for parallelization. Parallelism allows multiple threads of execution to run concurrently within a single program or process. OpenMP uses a fork-join model, where a master thread creates a team of threads, each executing a copy of the same code in parallel. The parallelism is achieved through the use of compiler directives, which are added to the code and indicate which parts of the program can be executed in parallel. OpenMP provides a range of constructs for creating, synchronizing, and coordinating threads.
To define a parallel region the directive **#pragma** omp parallel is used. OpenMP defines directives to enable parallelism within for

loops as well as nested for loops. These were used in the implementation and could significantly improve execution time.

OpenMP for performance measurements. OpenMP provides several tools and techniques that can be used to measure performance. To analyze the performance of the written code, the function omp_get_wtime() is provided. When used at two different points, the difference can be computed to get the execution time for a specific code section. This was used within the project, to identify bottlenecks and the effects of specific performance improvement methods.

Guided-vectorization with OpenMP. Vectorization is a technique that allows multiple data items to be processed simultaneously by a single instruction, which can significantly improve the performance of certain types of computations.
OpenMP provides several constructs for vectorization, including the "simd" directive, which instructs the compiler to generate SIMD (Single Instruction, Multiple Data) instructions for a loop.
To use the "simd" directive, it is important to ensure that the loop body can be vectorized. This typically involves ensuring that the loop is memory-bound rather than compute-bound and that the data access patterns are regular and aligned. Once the loop is vectorized, the compiler can generate optimized code that uses SIMD instructions to process multiple data items in parallel.

Cache-friendly code. To improve performance it is also important to know about memory and cache. To fully utilize the cache several things can be considered including spatial locality, temporal locality, and considering the size of the cache and cache lines.

3 EXPERIMENTS

3.1 Performance measurements

The implementation makes use of performance improvement measurements like parallelism, vectorization, and writing cache-friendly code. The effect of these optimization techniques is analyzed by debugging and measuring the execution time of the specific sections.

Table 1: Comparison of the run time with and without the use of OpenMP directives. The directives enable parallel execution for normal and nested for loops. The table shows the specific code section and the respective execution time in seconds. The sections measured were measured with functions provided by OpenMP.

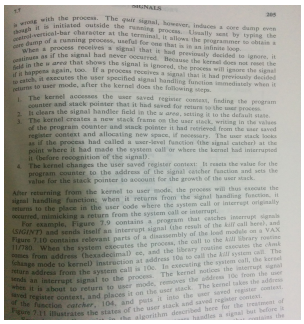
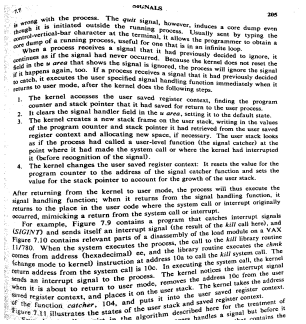
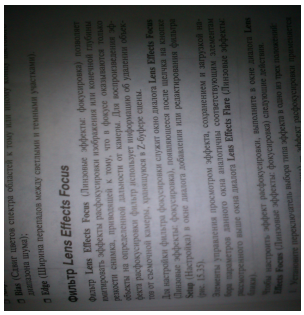
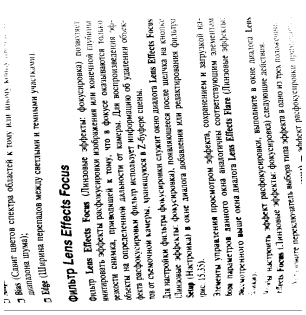
Description	Running time (s)
Median filtering with OpenMP	0.027506
Median filtering without OpenMP	0.085757
Adaptive thresholding with OpenMP	0.59191
Adaptive thresholding without OpenMP	1.83864
Conversion PPM/PGM with OpenMP	0.009831
Conversion PPM/PGM without OpenMP	0.015723

The run time was measured for different input images. This showed different performance measurements depending on the input images as well as the size of the image. However, for all images, using OpenMP directives lead to significant performance improvements.

3.2 Sample images

To analyze the effectiveness of the image enhancement, a set of images has been used. The input and output images can be compared to examine the effect of image processing. Table 2 shows some of the results.

Table 2: Comparison of input and output images. The input image is the one provided to the implementation and the output image is the one produced after image processing.

Input	Output
	
	

The implementation improves readability. Nevertheless, some improvements could be made. The filter size of the median filter was adjusted from a 3x3 to a 2x2 sliding window. This was done to improve readability. With a 3x3 window size, the text borders became unclear. With a 2x2 window, this effect is reduced.

4 CONCLUSIONS

In conclusion, the implementation of the enhancer for scanned images shows improvements in image quality, particularly in terms of text readability. It produces an image that minimizes artifacts and noise.
Furthermore, the implementation makes use of efficient methods to improve and optimize performance. The effect of these methods can be seen in the measurements made.
Further research and development can be made to improve the enhancer. Using another filter could be considered, as the borders of the text get unclear with increasing window size. Additionally, the

adaptive mean thresholding method could be adjusted to support PPM images and avoid the conversion made.

REFERENCES

[1] Sushant Pawar, Prasad S Halgaonkar, JW Bakal, and VM Wadhai. Implementation of ppm image processing and median filtering. *International Journal of Computer Applications*, 975:8887, 2011.

[2] Payel Roy, Saurab Dutta, Nilanjan Dey, Goutami Dey, Sayan Chakraborty, and Ruben Ray. Adaptive thresholding: A comparative study. In *2014 International conference on control, Instrumentation, communication and Computational Technologies (ICCICCT)*, pages 1182–1186. IEEE, 2014.

[3] T. Romen Singh, Sudipta Roy, O. Imocha Singh, Tejmani Sinam, and Kh. Manglem Singh. A new local adaptive thresholding technique in binarization. 2012.