Optimized Image Enhancer

Kenny Gozali
Friedrich Schiller University Jena
Germany
kenny.gozali@uni-jena.de

Walter Ehrenberger
Friedrich Schiller University Jena
Germany
walter.ehrenberger@uni-jena.de

Abstract

We report the development of an optimized image enhancer for files in the ppm format. This tool is created to make photographed text readable by enhancing said image through various filters. Our goal was to optimize the process by first, only using built-in tools that are available everywhere and second, using the gained control to optimize the process mainly using OpenMP. The following paper will try to give insights into the process, used methodology as well as discussing the results.

Keywords: image processing, vectorization, Portable Pixmap, benchmarking

ACM Reference Format:
Kenny Gozali and Walter Ehrenberger. 2023. Optimized Image Enhancer. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Image processing is a massive field in computer science. Through rigorous attempts, the techniques used in image processing improved throughout the years, especially due to the pursuit of advancing artificial intelligence in the field of computer vision [3]. In this paper, we have performed different techniques of image processing on an image using filters that will be discussed later on.

Every step performed on taking an image can introduce noise. From the digital process itself, to the available hardware as well as the environment in which the image is being taken (for example on sunset). Due to this reason, it is necessary to use various filters and techniques to deal with the noise adequately. Here, we especially focus on making hardly readable text on an image readable by using said filters. However, these filters are expensive in respect to the time complexity, even more so when multiple of them are chained together. One filter must loop at least once through every pixel, however every pixel has 3 values and some filters must loop over the pixel matrix multiple times.

This introduces the need to make improvements on the filters, which is done here by utilizing modern hardware that takes advantages of multiple threads and cores, as well as a statically typed language that operates close on said hardware. The most obvious approach is to collapse the previously defined loops, for them to be used in parallel. This approach works only as long as there are no direct data dependencies or the need for sequentiality like in a file stream. Vectorization is another important technique that allows a single instruction to operate on multiple data elements simultaneously. But here we also have to deal with data dependencies and ensure contiguous memory locations as well as other restrictions [4].

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Background

This section describes the necessary background knowledge for the following chapters.

2.1 PPM Format

In this project, we used the Portable Pixel Map (PPM) format. Even though, this format is highly inefficient, because it is redundant and contains information that is not even visible to the human eye. However, the simplicity of the specification makes it a useful format to start with [1].

There are two parts to the specification, the header, as well as the pixel matrix itself. The header contains only four pieces of information, but can contain a comment with a leading # at any place and multiple times. The information includes the mode which also ensures that the image is in the ppm format, image width, image height and the maximum value in the pixel matrix. The matrix itself is a raster of height rows from the top to the bottom, where each row consists of the image's width times 3 number of pixels. Times 3 because every pixel consists of 3 values for red, green and blue. All the information is encoded in ASCII values.

2.2 OpenMP

OpenMP is an open source API made for the purpose of parallel computation. The multiprocessing capability of the CPU is utilized to achieve higher performance. It is available for various programming languages, such as C/C++ and Fortran. OpenMP is capable of interfacing with various Instruction Set Architecture(ISA) and Operating Systems (OS), with the

intent that it will be a portable API. It takes advantage of the fork-join threading scheme.

The fork-join threading scheme works by separating the parallel process in 2 parts, the parallel region and the sequential region. In the sequential region, there exists only a single master thread that executes a sequential process. Whereas in the parallel region, the thread splits up into parallel threads and executes a parallel routine.

Vectorization is a technique that takes advantage of the Single Instruction Multiple Data (SIMD) parallel processing type [2]. There are various levels of vectorization, auto vectorization, guided vectorization and low-level vectorization. Auto vectorization requires only to add compiler flags, and it will work on certain pre-conditions and the programmer has less control over them. Guided vectorization works by taking advantage of an API such as OpenMP. In guided vectorization, the programmer is able to have more control. Low-level vectorization refers to using vector intrinsics to deploy vectorizations. In this case, the instructed set based on the CPU architecture is accessed directly. We decided to focus on guided vectorization to ensure the portability of the project. Within this project, for each filtering operation, each pixel is assigned a thread.

3 Process

In this project, we have managed to enhance images to improve their quality and readability. This enables us to read a low quality image of a text and make it readable. The steps, that are taken to achieve this goal, are explained in the following. First, an image with a ppm extension is taken, and the pixel matrix gets extracted. On this matrix we apply various filters. Lastly, a higher quality image will be obtained by stitching the obtained matrix back together with the original header information.

3.1 PPM Converter

There are existing libraries that solve this issue, however using these libraries means giving up control over the implementation and optimization. On the contrary,, this means any feature must be implemented by hand, which restricts the scope. For that reason, we only support ppm files with the mode P6 for now, as they are faster and more compact compared to the P3 mode. Implementation wise the only difference is that the image is in text form, in the P3 format while it is in a binary format, in P6. On a given image, the header information is extracted and saved while comments are eaten (Note that there remains a bug that multi line comments make the image invalid). After extraction of the meta information, a 2D vector-matrix is allocated which will be passed along to the filters in the following. Given the width times 3 and the height, this matrix is filled with the values of the byte stream of the image's pixel values. The values must be cast into an unsigned char first, to comply with the ASCII characters, then it can be cast into an integer which is necessary for performing processing. We tried two methods to optimize the pixel extraction. First, by collapsing the two outer loops that run through the pixel matrix. This results in an approximated two times speed up. Second, by vectorizing the array for storing the three pixel values, however this does not seem to have any meaningful impact using an Intel i5-4690 CPU. We found no methods to optimize the exportation of the pixel matrix, as the byte stream had to be accessed sequentially. Though this could possibly be improved by also using a buffer.

3.2 Filters

There are plenty of filters and techniques that are prevalent in the current literature. Image filtering enables the possibility of discovering new insights from an image, while at the same time decreasing or removing undesirable features such as noise. One of the most often used filters are for example the Gaussian filter and the median filter.

The image is comprised of vectors of vectors. requires to be padded in order to apply filter to them. There are various type of padding that can be used, such as zero-padding or boundary padding method. In this project, the zero-padding is used out of convenience.

3.2.1 Convolution. Convolution is a method used to take advantage of the neighbouring pixel. Through convolution, it is possible to do various filters and operations on the image.

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=\infty}^{\infty} \sum_{i=\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$

3.2.2 Gaussian Filter. The Gaussian filter applies a weighted average to the pixels in an image. It induces a smoothing effect, removes edges artifacts, and thus improves the overall quality.

3.2.3 Median Filter. The median filter is non-linear. It is highly effective against salt-and-pepper noise. It takes the neighbouring pixel, sorts the pixels, and then returns the median.

3.2.4 First Custom Filter - Contrast. This custom filter takes in the 2D pixel matrix, applies a kernel convolution operation, adjusts the contrast, and returns the filtered image. It then loops over each pixel in the image, skipping the pixels on the edge since the kernel would not fit, and applies a convolution operation using a 3x3 kernel. After the convolution operation, the pixel values are adjusted by a factor of contrast and clamped between the minimum value of 0 and the maximum value of 255. The performed optimization is a collapsing of the two outer loops.

3.2.5 Second Custom Filter - Extreme Mode. The second custom filter takes advantage of multiple layers of filters. The first layer will be the Gaussian filter. afterward, it will

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 1. The Gaussian Kernel

take the blurred image output from the Gaussian filter and use it to some extent as a divisor to the original image. The filtered image is then gamma corrected, in order to improve its sharpness.

3.3 Tooling: CMake

In software development, CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method. CMake is not a build system, but rather it generates another system's build files. It supports directory hierarchies and applications that depend on multiple libraries.

4 Result

This section outlines the findings from our project. Including the benchmark and the resulting output image.

4.1 Benchmark

The benchmark is done on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz. It has four cores and two threads within each core, which enable hyper-threading. And on an Intel(R) Core(TM) i5-4690 CPU @3.5 GHz, also with 4 cores. The results indicate an increase up to approximately 3.5 times from around 3.5 seconds to 1.0 seconds on both CPU's, as can be seen in 2 and in 3. As one can see, the increase in performance is capped after 4 threads, which leaves room for improvement. In the following table 1 one can see the improvements from one two four threads depending on the current step of the code.

4.2 Resulting Output

Using the explained methodologies, we can export two different kinds of output images. The original image can be seen here 4. And the first enhanced version can be seen here 5. It displays slightly clearer text that makes distinguishing the different letters an improvement for the reader. A second version of the enhanced image can be seen here 5. It is an extreme variant that might only be useful for extremely dark conditions, as the contrast is very high.

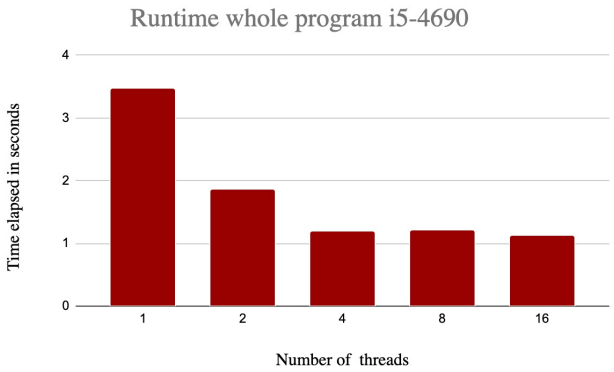

Figure 2. Performance increase by number of threads using an i5-4690 on a 1562x1200 image.

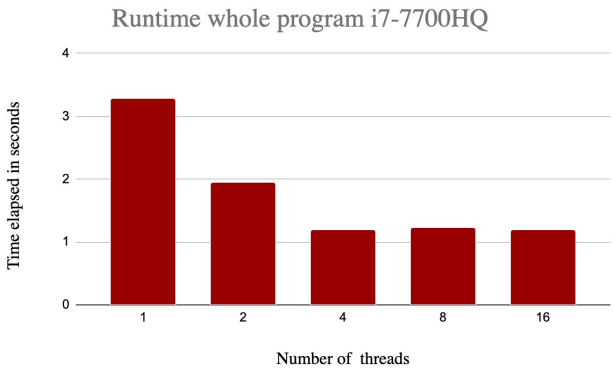

Figure 3. Performance increase by number of threads using an i5-7700HQ on a 1562x1200 image.

Step	1 Thread	4 Threads
MetaData extraction	0.00011	0.00011
Pixel extraction	0.22	0.081
Gaussian Filter	0.46	0.22
Median Filter	2.008	0.58
Custom Filter Extreme	0.4	0.12
Exporting PPM	0.088	0.098
Whole program	3.27	1.19

Table 1. Performance increase from one to four threads for every step in the program in seconds.

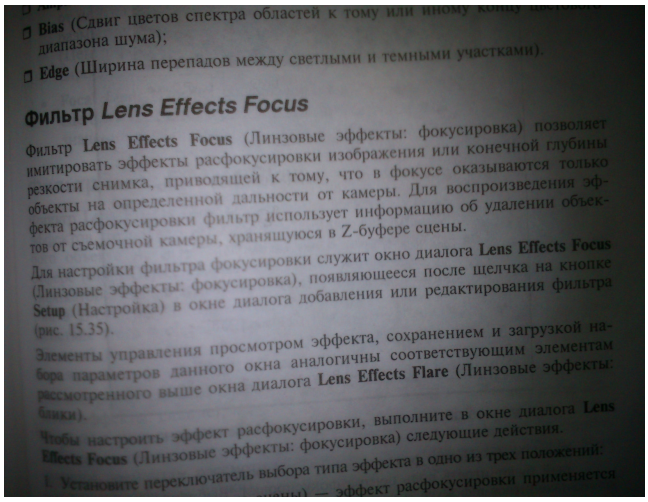

Figure 4. The original image that needs to be enhanced.

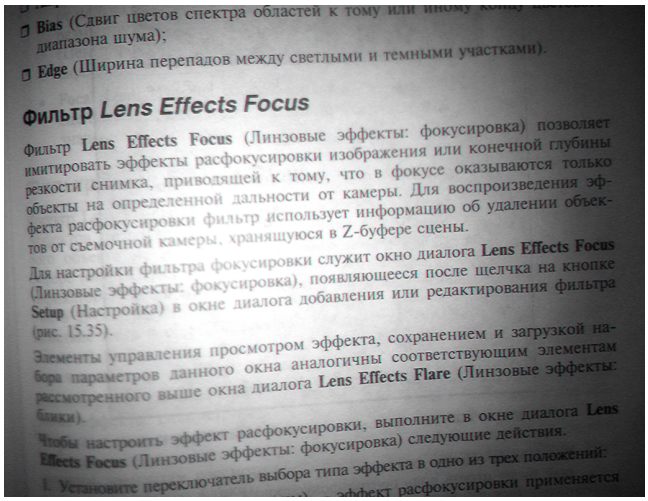

Figure 5. Enhanced version of the original image.

5 Discussion

The results in this project have to be viewed critically. Though there is an improvement to the execution time, the performance increase only works with up to four threads on the given architectures. And some of the vectorization methods resulted in less of an improvement than initially anticipated. The resulting image, though clearer to read than the original image, can also be improved upon further, as some of the text is darker, especially on the edges of the image. It could probably have been possible to implement a less intense version of the extreme filter, which mixed with the normal filter could have resulted in an improved image. These steps were tried but after initial failures not further pursued.

This being said, the project has showed us the possibility of employing optimization through parallelization and vectorization, and even with small tweaks to the code there is a

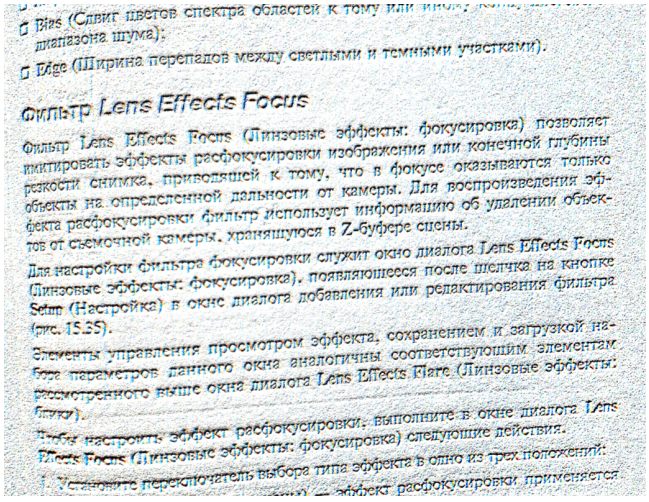

Figure 6. Extreme mode from our library, utilized on the original image.

significant increase in performance. Other steps to improve upon the project would be the usage of profiling tools such as Valgrind or Intel One API. It is imperative that profiling is done within a project, because it would give us an insight on the source of the bottlenecks. The various filters and image enhancement techniques in this paper were useful, however there are definitely rooms for improvement. There are different image enhancing and filtering tools that employ various different techniques, such as deep learning or histogram equalization. Due to limited resources, we were only able to test this project using two different CPU's. One suggestion that came from us would be to use more variations of CPU's with different number of cores and to test it on different number of threads. Perhaps in the future it is also possible to do the implementation using Non-Uniform Memory Access and a different parallel computing API, for instance Message Passing Interface (MPI).

References

[1] 2016. *PPM Specification.* Retrieved March 14, 2023 from https://netpbm.sourceforge.net/doc/ppm.html

[2] Michael J. Flynn. 1972. Some computer organizations and their effectiveness. *IEEE Trans. Comput.* C-21, 9 (1972), 948–960. https://doi.org/10.1109/tc.1972.5009071

[3] David A. Forsyth and Jean Ponce. 2015. *Computer vision: A modern approach: A modern approach.* Pearson Education.

[4] Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. 2020. There's plenty of room at the top: What will drive computer performance after moore's law? *Science* 368, 6495 (2020). https://doi.org/10.1126/science.aam9744