# Enhancer for Scanned Images Using Parallelized Image Binarization Methods

**Algorithm Engineering 2025 Project Paper**

Salim Alkhaddoor
Friedrich Schiller University Jena
Germany
salim.alkhaddoor@uni-jena.de

Alexander Keller
Friedrich Schiller University Jena
Germany
alexander.keller@uni-jena.de

## ABSTRACT

When digitizing large amounts of physical documents, it is important to process images in a way that is memory efficient, fast and maintains the integrity of all relevant information at hand. This work presents parallelized implementations of two state of the art image binarization algorithms (NICK and Sauvola) and compares them to non-parallelized implementations. Additionally, the impact of integral images is evaluated. We focus on the running time of the different implementations, in particular, what improvement is to be expected when making use of multiple CPU cores. We were able to improve the time efficiency of the different methods greatly by making use of simple directives using OpenMP and by using the aforementioned Integral images. These findings show the importance of paying special attention to the computational resources available and of making use of them, whenever possible.

## KEYWORDS

image binarization, thresholding, parallel processing, efficient computing

## 1 INTRODUCTION

Currently in numerous institutions internationally, data is still stored physically, i.e. on paper. While the world shifts from physical towards digital media, the same holds true for bureaucratic endeavors. The challenge of transferring data from paper documents onto digital hard drives arose, where a number of demands have to be met and certain constraints have to be adhered to.

The disk space necessary to store the documents has to be reasonable, where reasonable may depend on a number of factors such as type of document and importance of the data present (e.g. no compression at all may be desirable for historic preservation). To reach that goal the images are often times stored as *binary images*, which are images where the intensity values of all pixels in the image are limited to only two values (for example 0 and 1). This allows for highly effective compression at the cost of the image content obviously being altered.

The second requirement is a high degree of accuracy in the transfer process. Ideally, all important information should be kept when digitizing, or more specifically in our case, binarizing an image. When assuming only textual content in the image the problem turns into the problem of distinguishing the background (paper) from the foreground (text) and assigning the value of 1 only to the text. For this purpose, there are a number of different methods. A simple thresholding algorithm and two state of the art methods are implemented in our repository (github.com/SalimSama/Algorithm-Engineering-Course), which are NICK [5] and Sauvola [6]. Both methods deliver satisfactory results (see Figure 2).

Lastly the process should ideally run in a short amount of time. Especially when considering highly detailed images with thin pen strokes or when using elaborate methods of compression or binarization, this may become increasingly important. With international documents in foreign and sometimes very elaborate handwriting and print being processed daily, an efficient solution that

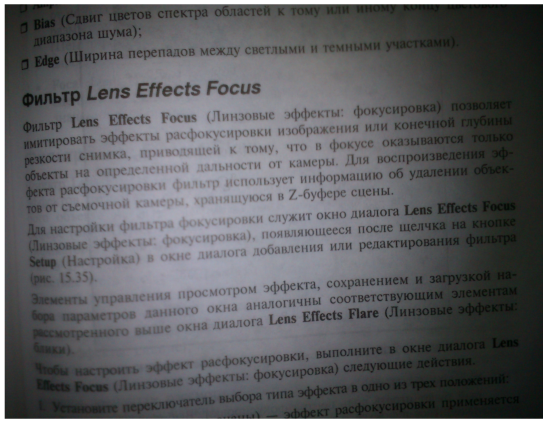


**Figure 1: Original image with Cyrillic writing, unfavorable lighting conditions and noise.**

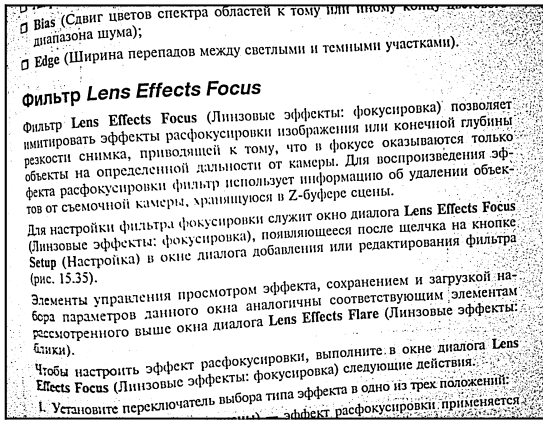


**Figure 2: Resulting image using Sauvola's method.**

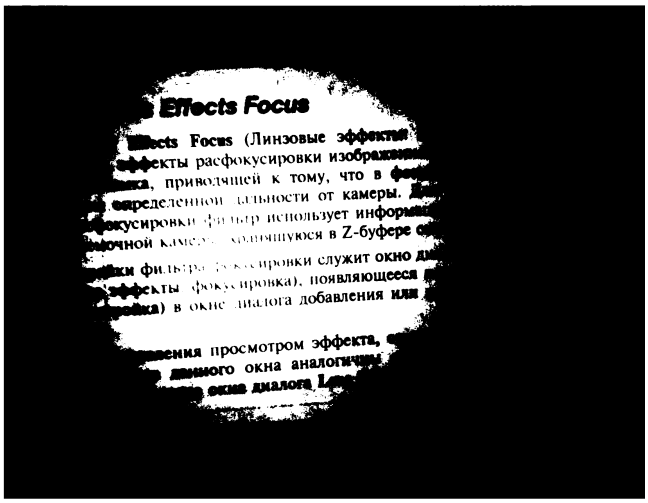

**Figure 3: Resulting image using global binarization.**

makes use of the computational resources is necessary and is the focus of our analysis over the course of this work.

The illustrative example of digitizing images in various institutions is not the only case in which binarization is useful. The same process and requirements can be applied in different scenarios, i.e. as a pre-processing step for machine learning algorithms or for text recognition purposes. Real-time applications such as augmented reality software may require image binarization in a pre-processing step as well. To deliver a good result, a fast running time is necessary, which is an example of why we require software which makes use of the hardware available.

## 1.1 Background

The first binarization method we implemented is a simple *global thresholding* algorithm [1], which can be represented by a transformation $T_c(r)$ of an input pixel $r$ with

$$T(r) = \begin{cases} 1 & r \leq c, \\ 0 & \text{else} \end{cases}$$

and threshold $c$. Global thresholding methods determine a single threshold to apply to the entirety of the image (meaning each pixel). These approaches are easy to implement and very efficient, but are not particularly robust since they are susceptible to noise, reflections and changes in lighting conditions. Another class of thresholding algorithms are *local thresholding* algorithms. These algorithms determine a new threshold $c$ for each pixel or region by taking local properties into account [1]. The two local thresholding methods we implemented are NICK and Sauvola. Both produce good results over a wide range of documents [2] and follow a similar principle of using the image statistics within a window around the current pixel being processed.

Let $\mu$ be the mean and $\sigma$ be the standard deviation of intensity values of pixels within a square section of the image. We will call such a section *a window*. For **Sauvola's method** the equation for

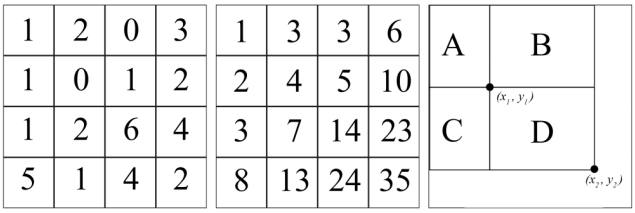


**Figure 4: A two dimensional intensity image, with a corresponding integral image and a representation of how any square area (D) can be calculated.**

determining the threshold $c$ for each pixel is the following:

$$c = \mu(1 - k(1 - \frac{\sigma}{R})),$$

with $R$ being the predetermined gray level value and $k$ being a control factor within $[0.2, 0.5]$. Considering images with gray level value within $[0, 255]$, the values suggested by the author are $k = 0.2$ and $R = 125$ [6]. The **NICK method** (as proposed by **N**icole Vincent, **I**mran Siddiqi, **C**laudie Faure, **K**hurram Khurshid) calculates the threshold as follows:

$$c = \mu + k\sqrt{\frac{\sum_{i \in I} p_i^2 - \mu^2}{NP}},$$

where $k \in [-0, 1, -0.2]$, $p_i$ is the gray scale value of the pixel ($I$ designating the pixel positions within the window) and $NP$ is the number of pixels [5].

The running time of these methods can be sped up considerably by making use of *integral images* [4]. The integral image $I$ of an image $f$ stores in the location $(x, y)$ the sum of the values of $f$ to the left and above $(x, y)$:

$$I(x, y) = \sum_{x=1}^{n} \sum_{y=1}^{m} f(x, y)$$

The sum of all pixels within a given window can be calculated using integral images in constant time: Given two positions, $(x_1, y_1)$ making up the upper left and $(x_2, y_2)$ making up the bottom right corner of a window. The sum of all values inside of that window is

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x, y) = I(x_2, y_2) - I(x_2, y_1) - I(x_1, y_2) + I(x_1, y_1).$$

We will use this to our advantage to improve our implementations.

## 1.2 Related Work

Our work is mainly based on two papers, which detail the methods we implemented [2] and how integral images could be used to improve the running time [4].

## 1.3 Our Contributions

Since the goal of our implementation is a short running time, we used the C++ programming language and utilized beneficial compiler directives. The methods are implemented to perform the computations on multiple CPU cores, if they are available. We have done this for a simple global thresholding algorithm with a user defined global threshold. The API facilitating this is OpenMP [3]. The

aforementioned integral images are used to make further improvements, which is used in combination with the OpenMP directives whenever appropriate. We compared the different implementations in terms of the performance advantage they provided and compared the resulting binarized images to confirm our implementation's correctness.

## 2 THE IMPLEMENTATION OF THE BINARIZATION ALGORITHMS

### 2.1 Global Binarization

The first algorithm that is implemented is a simple global thresholding algorithm with a user defined threshold. The non-parallelized version of this algorithm iterates over all pixels in the image and changes them to 1 or 0 depending on whether the intensity value is larger or smaller than the threshold. This was improved by adding the `#pragma omp for` directive before the for-loop. Further improvements were achieved by using the construct `simd` after the directive to indicate, that multiple iterations can be executed concurrently using SIMD (Single instruction, multiple data) instructions.

### 2.2 Sauvola and NICK

The parallelization of our implementations of Sauvola and NICK were done in a similar fashion, also making use of the directives offered by OpenMP. Since we iterate over each pixel, we can once more utilize the `#pragma omp for` directive.

### 2.3 Integral Image

We attempted to improve the running time further by making use of integral images. Before calculating the image statistics, we calculate the integral image, which can be computed in linear time. We achieve this by first iterating over the image row-wise, where in each entry the sum of intensity values to the left (including the intensity value of the entry itself) is stored and then iterating column-wise over the resulting image in the same way. This procedure follows directly from the definition of integral images, where we apply a dynamic programming approach in order to avoid calculating the same value multiple times.

The calculation itself was also parallelized, by having each thread compute the values for one row or column at a time. The resulting integral image can then be used to calculate the sum of all intensity values within a window in constant time by exploiting the method described in section 1.1.

## 3 EXPERIMENTS

We benchmarked our implementation on a Linux machine running the Fedora Linux 41 distribution. The benchmark ran on a laptop with a 12th Generation Intel i5 (2,80 GHz 12 Threads) processor and 16GB of RAM. The tests were performed five times, averaged and then compared against an unparallelized implementation. Figure 5 and 6 show that the we achieved considerable running time improvements using the methods specified. All implementations benefit from an increasing amount of threads and show a speedup that seems to be linear with the amount of threads, although we suspect that the benefit will diminish as the amount of threads increases. We can see that the parallelization with OpenMP was very

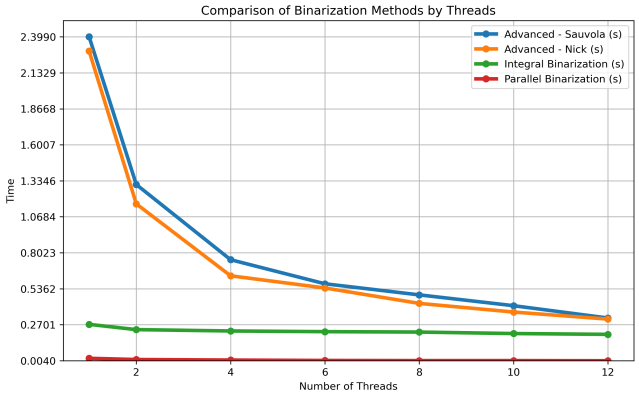


**Figure 5: Comparison of the running time of the different implementations and rising amount of threads.**

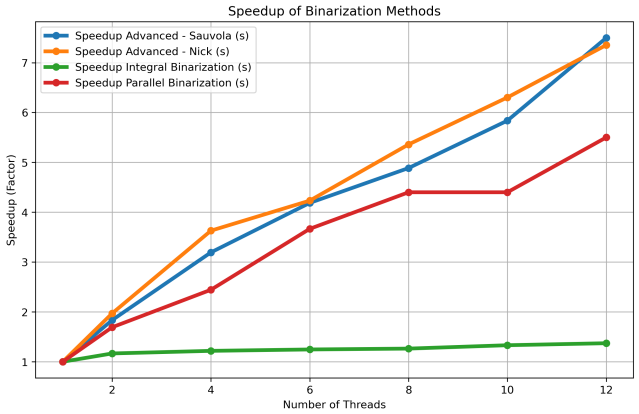


**Figure 6: Speedup achieved by implementation and number of threads used.**

effective, distributing the work among all cores available. The SIMD instructions offered improvements, speeding up the computation by performing multiple instructions in a single clock cycle.

In terms of storage space, the binarization of images show dramatic improvements. The image displayed in figure 1 requires a storage space of 5.652KB while the binarized version in figure 2 requires only 220KB, reducing the storage size by a factor of roughly 27. When evaluating the compression rate for a series of other images, we also observed considerable improvements.

## 4 CONCLUSIONS

It is important to use the resources available when attempting to write efficient code. This does not have to be egregiously complicated since parallelization can be made very straight forward using different APIs. Another approach of gaining performance improvements is on the algorithmic side. Making use of different representations (in our case Integral images) can induce an advantage when performing different kinds of computations, therefore the improvements are not only to be looked for in the utilization

of all resources or increase of such resources, but also in the algorithmic or representative side of things. The resulting images very clearly confirm what has been stated by Bataineh et al. [2], which is that Sauvola and NICK are very useful and robust binarization methods and that global thresholding methods tend to produce rather weak results 3. To improve the user experience, one could implement a pipeline of methods, where different filters are applied in succession or multiple images are processed one after the other.

## REFERENCES

[1] Nafiz Arica and Fatos Yarman Vural. 2001. An overview of character recognition focused on off-line handwriting. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 31 (06 2001), 216 – 233. https://doi.org/10.1109/5326.941845

[2] Bilal Bataineh, Siti N. H. S. Abdullah, K. Omar, and M. Faidzul. 2011. Adaptive Thresholding Methods for Documents Image Binarization. In *Pattern Recognition*, José Francisco Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, Cherif Ben-Youssef Brants, and Edwin Robert Hancock (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 230–239.

[3] OpenMP Architecture Review Board. 1997. OpenMP Homepage. www.openmp.org. Accessed: 2025-01-28.

[4] Derek Bradley and Gerhard Roth. 2007. Adaptive Thresholding using the Integral Image. *J. Graphics Tools* 12 (01 2007), 13–21. https://doi.org/10.1080/2151237X.2007.10129236

[5] Khurram Khurshid, Imran Siddiqi, Claudie Faure, and Nicole Vincent. 2009. Comparison of Niblack inspired Binarization Methods for Ancient Documents. 1–10. https://doi.org/10.1117/12.805827

[6] Jaakko Sauvola, Tapio Seppänen, Sami Haapakoski, and Matti Pietikäinen. 1997. Adaptive Document Binarization. *Pattern Recognition* 33, 147–152 vol.1. https://doi.org/10.1109/ICDAR.1997.619831