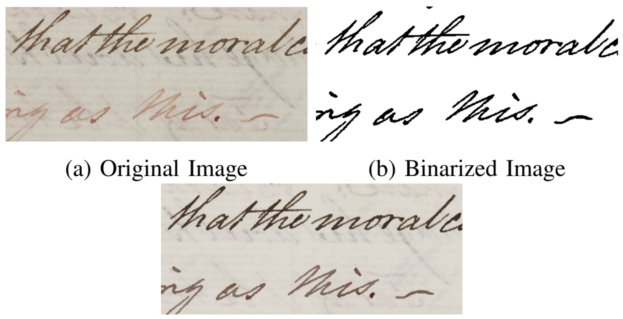
Efficient Binarization for Historical Document Analysis

Florian Westphal, Håkan Grahn and Niklas Lavesson
Department of Computer Science and Engineering
Blekinge Institute of Technology
Karlskrona, Sweden
Email: {flw, hgr, nla}@bth.se

***Abstract*—Readability of document images is one core issue when analysing historical documents. One way to improve the readability of those document images is image binarization. By separating the written text from its background, documents degraded by, e.g., stains or faded ink become better readable. Due to the large quantity of available historical document images, this binarization needs to be done efficiently to make this form of processing feasible.**

In this paper, we present our work in progress on improving the execution performance of a state-of-the-art binarization algorithm by mapping it onto a heterogenous platform. We describe how the algorithm can be divided and computed in parallel on CPU and GPU. The preliminary results, which we report in this paper, suggest that a speedup of 1.4 is possible in comparison to the original algorithm.


(a) Original Image (b) Binarized Image
(c) Merged Image
Fig. 1: Improved Readability through Binarization

I. Introduction

Historical handwritten documents have been photographed and archived on micro film for several decades. More recently, the possibility to digitize such documents has made them available to a broader audience. Our partner ArkivDigital[1], for instance, provides genealogists with access to around 50 million images of Swedish church records and other historical documents. Despite the good image quality, images of old documents are still hard to read, due to, e.g., faded ink, ink bleeding through from the other side of the page, or stains.

One way to improve the readability of these documents is to use image binarization. This technique separates the written text in an image from its background by classifying pixels of the image as foreground or background pixels, depending on their characteristics in the image. The obtained binarized image can then be used to overlay the original image to improve its readability. Figure 1 illustrates an ideal case based on ground truth data from the DIBCO 2013 dataset [1].

While binarization algorithms must clearly separate the background and foreground of an image, they also have to perform this binarization in a timely manner. This need is derived from the vast amount of available images, which should be visually enhanced. In our case, we are looking at 50 million high resolution images. If the processing of one image takes only 30 seconds, then processing all images would take up to 48 years. Therefore, it is vital to use binarization algorithms, which produce good binarization results fast. The required speedup of these algorithms can be achieved by efficiently using the possibilities for parallel computation provided by heterogeneous platforms, which are widely available nowadays. A heterogeneous platform, is simply a system combining different processor types [2]. Those processor types are normally multicore central processing units (CPUs), integrated graphics processors (IGPs), and graphics processing units (GPUs).

In our work, we explore how a large number of document images can be binarized in a timely manner, to make it feasible to process all of ArkivDigital's images. To achieve this goal, we propose a processing pipeline in which image preprocessing is handled by the CPU, while the actual binarization is performed by the GPU. In this way, we utilize the heterogeneous platform to speed up Howe's binarization algorithm [3]. This processing pipeline distinguishes our approach from previous work, which used only the GPU to speed up binarization and considered only the binarization of one image.

In the following, we introduce Howe's binarization algorithm, as well as challenges of algorithm development for a heterogeneous platform in Section II. Section III describes other approaches, which attempt to speed up image binarization and identifies the difference between those approaches and our binarization pipeline. We describe our approach in more detail in Section IV and Section V states our experimental design. We present our preliminary results in Section VI and conclude this paper in Section VII.

[1]http://www.arkivdigital.net

II. Background

In this section, we introduce image binarization in general and describe Howe's binarization algorithm, on which our approach is based. Additionally, we describe the challenges of mapping such an algorithm to a heterogeneous platform and motivate why the use of multiple computing devices does not guaranty faster execution.

A. *Image Binarization*

Image binarization is the process of classifying image pixels as foreground or background pixels. For images of handwritten documents, this means to separate the written text from its background. This is especially challenging when dealing with images of historical documents, which can be degraded by, for instance, faded ink, stains covering the written text, or ink bleeding through from the other side of the page. A binarization algorithm for historical documents has to be able to cope with these degradations, which increase the risk that foreground pixels are accidentally classified as background pixels or vice versa.

Binarization algorithms by Otsu [4], Niblack [5], and Sauvola et al. [6] are well known. These algorithms are commonly used as baseline for comparison with new algorithms, for instance, in competitions, such as the Document Image Binarization Contest (DIBCO) held at the International Conference on Document Analysis and Recognition (ICDAR) [7], [1] or the Competition on Handwritten Document Image Binarization (H-DIBCO) held at the International Conference on Frontiers in Handwriting Recognition (ICFHR) [8], [9]. The winner of the latest binarization contest in 2014 [9] was a combination of the algorithms by Howe [10] and Mesquita et al. [11]. Since the difference in binarization quality between Howe's algorithm and the winning binarization algorithm was small, but the winning algorithm was reported to require a substantial amount of time more, we decided to work with Howe's binarization algorithm instead.

Howe's binarization algorithm labels image pixels as foreground or background pixels by minimizing a global energy function. This function penalizes labelings that do not conform with the image's Laplacian, that is, it penalizes pixels in intensity valleys (ink), which are classified as background and pixels in intensity plateau or peak areas (background), which are classified as foreground. Additionally to this simple separation based on the divergence of the gradient, the energy function penalizes labelling discontinuities, unless they take place at an edge, as detected by the Canny edge detection algorithm [12]. This addition improves the stability of the binarization algorithm and encourages the continuity of foreground and background areas.

B. *Heterogeneous Computing*

As stated before, heterogeneous computing uses different processor types simultaneously to improve the execution speed of an algorithm. One simple way to achieve this is to distribute different parts of the algorithm to different processors. This allows two levels of parallelism, namely the parallel execution of those different parts on different processors and on the other hand the parallel execution of each of these parts on its respective processor.

Despite this good potential for increasing the execution speed, mapping an algorithm onto a heterogeneous platform entails many challenges. One of these challenges is memory management. While CPU and IGP usually both use the computer's main memory, the GPU possesses its own, smaller memory space. Moving data between those different memory spaces inflicts high performance penalties. If not considered carefully, this performance loss can outweigh the speedup gained by parallel execution. Another challenge is the diversity of execution models of the different processor types. This diversity makes it necessary to carefully select and optimize algorithms depending on which processor is executing them. Apart from these challenges, the code execution on different processor types has to be coordinated. One way to do this is to use the OpenCL standard [13]. This standard defines a unified programming interface, which allows it to program and coordinate the execution of programs on many different processor types.

III. Related Work

Most studies on image binarization focus on the quality of their binarization result without considering the required execution time. One typical example for this is the work by Nina et al. [14]. However, there are studies, such as the one by Soua et al. [15], which use the GPU to speed up image binarization. In contrast to these studies, we use the GPU and the CPU of a heterogeneous system to speed up binarization and focus on the processing of large numbers of images.

IV. Binarization Pipeline

Section IV-A describes how we split Howe's binarization algorithm into smaller parts to be executed on different processor types. Their distribution and the pipelined processing of images is described in Section IV-B.

A. *Algorithm Decomposition*

Howe's binarization algorithm can be divided into three main steps. These steps are 1.) Laplacian computation, 2.) Canny edge image computation, and 3.) energy minimization, as shown in Figure 2. For our binarization pipeline, we divide the binarization algorithm into these three different parts. This is a reasonable separation, since any of these three steps can be computed independently, provided that the initially required data is available. Additionally, the amount of data that needs to be transferred between different memory spaces is reasonably small in this setting, since only the grayscale image, the Laplacian image, the Canny edge image, and the binarized image have to be exchanged.

B. *Algorithm Mapping*

This separation makes it possible to execute different parts of the algorithm on either CPU or GPU. Table I shows for the 3 different steps how they could be mapped onto CPU and

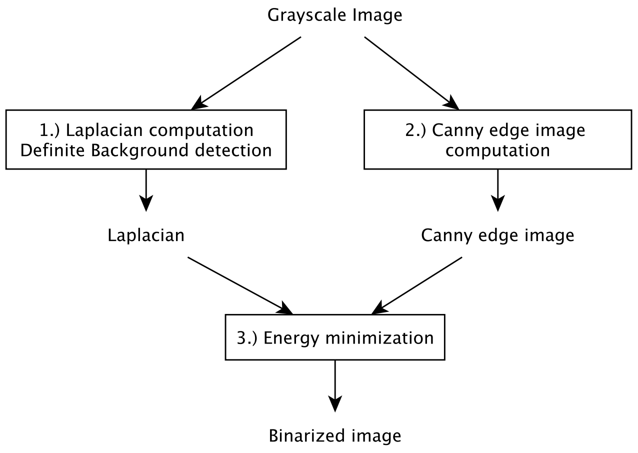

Grayscale Image
1.) Laplacian computation
Definite Background detection
2.) Canny edge image computation
Laplacian
Canny edge image
3.) Energy minimization
Binarized image

Fig. 2: Decomposition of Howe's Binarization Algorithm

GPU. In our work, we evaluate which of these 8 combinations produces the fastest binarization result.

TABLE I: Mapping of Algorithm Parts to CPU and GPU

	I	II	III	IV	V	VI	VII	VIII
1	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
2	CPU	CPU	GPU	GPU	CPU	CPU	GPU	GPU
3	CPU	CPU	CPU	CPU	GPU	GPU	GPU	GPU

As stated before, the algorithms used to compute each of these three steps have to be chosen carefully, based on the processor type on which they are executed. The best example for this is the energy minimization step. In his paper, Howe uses Boykov and Kolmogorov's graph cut algorithm [16] to perform the minimization of the energy function [3]. While this is an efficient algorithm when executed on a CPU, it is not suitable for execution on a GPU due to its global tree data structure. A graph cut algorithm, which is more suitable for execution on a GPU is the JF-cut algorithm by Peng et al. [17]. This algorithm is based on a push-relabel approach, which allows parallel local updates, in contrast to sequential updates to a global data structure, making it therefore more suitable for the GPU. For this reason, we are using in our implementation these two different algorithms for CPU and GPU.

Apart from the suitability of different processor types for different tasks and the selection of suitable algorithms for the different processor types, the simultaneous use of different processor types yields another optimization possibility. Figure 2 shows that the first two steps of the algorithm only require the grayscale image, while the energy minimization can be executed only after these two steps. This allows the computation of the next image's Laplacian and Canny edge image concurrently with the previous image's energy minimization. Thus, a batch of images can be binarized in a pipeline, where the next image is preprocessed, while the previous image is still binarized. This reduces the computation time by overlapping the computations.

V. Experiment Design

This section describes our experiment design together with the different measurement objectives. Section V-A describes the platform on which our experiments were conducted, as well as the used datasets. The measures used to assess the binarization performance are described in Section V-B and the measurement of the execution performance is described in Section V-C.

A. Experiment Setup

All experiments were performed on a laptop with a 2,8 GHz Intel Core i7 CPU, 16 GB of main memory, and an NVIDIA GeForce GT 750M GPU with 2048 MB of RAM. OpenCL 1.2 on OS X was used to coordinate the execution of the binarization algorithm on CPU and GPU. For the experiments, we used our OpenCL implementation of Howe's binarization algorithm, as well as his reference implementation, which was written in MATLAB.

As test dataset, we used the images from the 2014 competition on handwritten document image binarization (H-DIBCO 2014) [9], as well as one randomly chosen high resolution image from ArkivDigital. While the sizes of the 10 competition images range from 775×460 pixels to 2675×1255 pixels, the ArkivDigital image has a size of 5184×3456 pixels.

B. Binarization Performance Measurement

In order to assess the binarization quality of the 8 analysed combinations, we compare the binarization results obtained using these combinations with the ground truth data. We use two different measures, the pseudo F-Measure (F_{ps}) and the Distance Reciprocal Distortion Metric (DRD), for this comparison. Those measures were used before to evaluate the binarization performance in different binarization competitions [1], [9]. In contrast to other measures, these measures weight misclassified pixels based on their location in the image to give a more realistic view on binarization quality from a human perspective. While it is desirable to achieve a high F_{ps} value, a lower DRD value indicates a better binarization result.

C. Execution Performance Measurement

For each combination, we measure the time to perform the binarization, excluding the time for loading and storing an image. This time measurement is consecutively repeated four times of which the first measurement is discarded to ensure equal measurement conditions. The remaining three measurements are used to compute the average execution time for the particular combination and image.

Additionally to measuring the time it takes to binarize one image, we also measure how much time it takes to compute each of the three steps. As for the total times, these measurements are repeated four times of which only the last three are used to compute the average time taken for each step.

VI. Preliminary Results and Analysis

By performing the experiments as described in the previous section, we obtained the preliminary results, which are reported in this section. The binarization performance of all 8 combinations is discussed in Section VI-A and the measured execution times are reported in Section VI-B.

A. *Binarization Performance*

As described before, the binarization performance is evaluated based on F_{ps} and DRD. Figure 3 shows the box plots for the average F_{ps} and DRD over all H-DIBCO 2014 images for each combination. Here, the different combinations were separated based on which processor type executed the energy minimization, since it was clear from the obtained results that the used energy minimization algorithm influenced the binarization quality strongly. The box plots with the label "**C" display the binarization performance of the combinations using the CPU for the energy minimization, while "**G" identifies the binarization performances of combinations using the GPU for the minimization step. The black lines mark the obtained binarization results using the reference implementation.

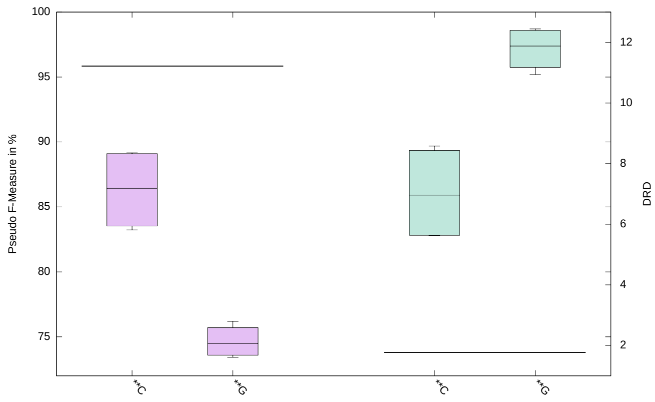


Fig. 3: Binarization Performance Separated by Processor Type for Energy Minimization

From Figure 3, one can see that the reference implementation performs clearly better than our implementation. Apart from that, the figure shows that our implementation does not produce consistent binarization results for all combinations. This is problematic, since the ideal case would be that all combinations yield the same binarization results, but differ in execution performance.

One possible reason for the inconsistent binarization results between combinations using the CPU for energy minimization and those using the GPU might be that sub optimal algorithm parameters were chosen for both groups. Apart from that, the difference between the reference implementation and the combinations using the CPU for energy minimization suggests that the preprocessing is not handled in an optimal way. This is the case, since the reference implementation uses the same implementation of Boykov and Kolmogorov's graph cut algorithm as the CPU energy minimization in our implementation.

B. *Execution Performance*

Figure 4 shows the accumulated time to binarize all images in the H-DIBCO 2014 dataset for each of the 8 combinations. From this figure, one can see that all combinations, except I and III, execute faster than the reference implementation, with combination VIII being the fastest. However, when binarizing the high resolution image, all combinations using the CPU for the energy minimization execute slower than the reference implementation, while the other four combinations perform faster. For this image, combination VI performs best, as shown in Figure 5.

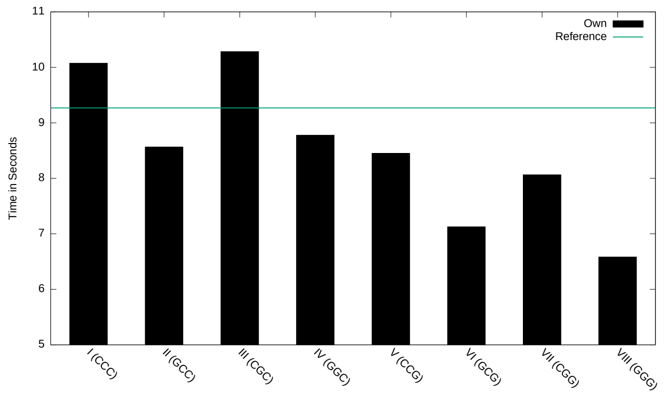


Fig. 4: Execution Performance of Each Combination for all H-DIBCO 2014 Images

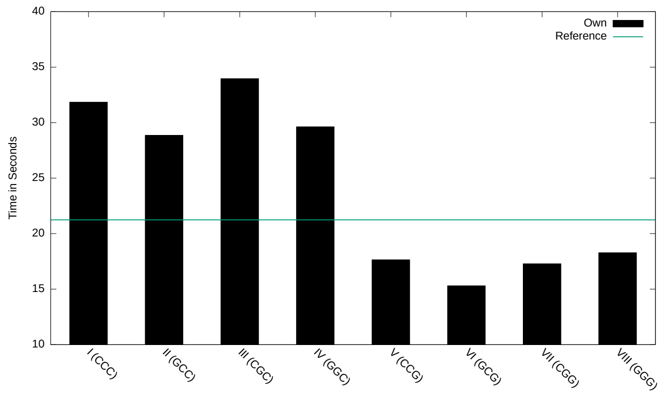


Fig. 5: Execution Performance of Each Combination for the High Resolution Image

Apart from increasing the execution performance by selecting efficient algorithms for the respective processor type and mapping the three steps, Laplacian, Canny, and energy minimization, to their most suitable processor type, one can also take advantage of the fact that different parts of the algorithm can be executed at the same time on different processor types. From Table II, it is apparent that a pipelined execution could speed up the binarization of an image by 0.5 seconds. This is the case, since the preprocessing on the CPU could be handled while the previous image is still binarized by the GPU. Therefore, the preprocessing steps would be already done when the GPU finishes with the binarization of the previous image and could directly move on to binarizing the next image. This could save 0.5 seconds compared to when preprocessing and energy minimization are executed on the GPU.

TABLE II: Separate Execution Times for the High Resolution Image

	1	2	3
CPU	2.27 s	0.17 s	28.76 s
GPU	0.39 s	0.11 s	14.54 s

VII. Conclusions and Future Work

In this paper, we have presented our general idea on how to reduce the execution time of a state-of-the-art algorithm, such as Howe's binarization algorithm with help of heterogeneous computing. We have described how this algorithm can be divided and mapped to a heterogeneous platform and how the possibility of parallel execution between CPU and GPU can be used to increase the binarization speed of large amounts of images by processing images in a pipelined fashion.

Additionally, we have presented our preliminary results, which show that the energy minimization is the most compute intense step and that the execution speed of this step can be improved by performing it on the GPU using JF-cut as graph cut algorithm. However, the analysis of the binarization quality has shown that the binarization performance is worse in those combinations, which use the GPU for energy minimization. Therefore, further investigation is required to understand the reasons for this reduced binarization performance. Possible reasons might be erroneous preprocessing or sub optimally chosen algorithm parameters.

Another issue requiring further investigation is the proposed binarization pipeline. While our preliminary results suggest that the expected speedup will be small, it would still be interesting to follow this idea further.

VIII. Acknowledgements

This work is part of the research project "Scalable resource-efficient systems for big data analytics" funded by the Knowledge Foundation (grant: 20140032) in Sweden.

References

[1] I. Pratikakis, B. Gatos, and K. Ntirogiannis, "ICDAR 2013 Document Image Binarization Contest (DIBCO 2013)," in *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2013, pp. 1471–1476.

[2] D. A. Patterson and J. L. Hennessy, *Computer organization and design*. Morgan Kaufmann, 2013, ch. C, p. 3.

[3] N. R. Howe, "A Laplacian Energy for Document Binarization," in *2011 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2011, pp. 6–10.

[4] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1979.

[5] W. Niblack, *An Introduction to Digital Image Processing*. Prentice-Hall, Englewood Cliffs, 1986, pp. 115–116.

[6] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern recognition*, vol. 33, no. 2, pp. 225–236, 2000.

[7] B. Gatos, K. Ntirogiannis, and I. Pratikakis, "DIBCO 2009: document image binarization contest," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 14, no. 1, pp. 35–44, 2011.

[8] I. Pratikakis, B. Gatos, and K. Ntirogiannis, "ICFHR 2012 competition on handwritten document image binarization (H-DIBCO 2012)," in *2012 International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2012, pp. 817–822.

[9] K. Ntirogiannis, B. Gatos, and I. Pratikakis, "ICFHR2014 Competition on Handwritten Document Image Binarization (H-DIBCO 2014)," in *2014 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2014, pp. 809–813.

[10] N. R. Howe, "Document binarization with automatic parameter tuning," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 16, no. 3, pp. 247–258, 2013.

[11] R. G. Mesquita, C. A. Mello, and L. Almeida, "A new thresholding algorithm for document images based on the perception of objects by distance," *Integrated Computer-Aided Engineering*, vol. 21, no. 2, pp. 133–146, 2014.

[12] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986.

[13] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in science & engineering*, vol. 12, no. 1-3, pp. 66–73, 2010.

[14] O. Nina, B. Morse, and W. Barrett, "A Recursive Otsu Thresholding Method for Scanned Document Binarization," in *Workshop on Applications of Computer Vision (WACV)*. IEEE, 2011, pp. 307–314.

[15] M. Soua, R. Kachouri, and M. Akil, "GPU parallel implementation of the new hybrid binarization based on Kmeans method (HBK)," *Journal of Real-Time Image Processing*, pp. 1–15, 2014.

[16] Y. Boykov and V. Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[17] Y. Peng, L. Chen, F.-X. Ou-Yang, W. Chen, and J.-H. Yong, "JF-Cut: A Parallel Graph Cut Approach for Large-Scale Image and Video," *IEEE Trans. on Image Processing*, vol. 24, no. 2, pp. 655–666, 2015.