Parallel Implementation of Niblack's Binarization Approach on CUDA

Brij Mohan Singh, Rahul Sharma
Department of CSE,
College of Engineering Roorkee,
Roorkee-247667,Uttarakhand, India

Ankush Mittal
Director (Research),
Graphic Era University, Dehradun-248002, Uttarakhand, India

Debashish Ghosh
Department of E&C,
IIT Roorkee,
Roorkee-247667,Uttarakhand, India

ABSTRACT
Image processing and pattern recognition algorithms take more time for execution on a single core processor. Graphics Processing Unit (GPU) is more popular now-a-days due to their speed, programmability, low cost and more inbuilt execution cores in it. Most of the researchers started work to use GPUs as a processing unit with a single core computer system to speedup execution of algorithms. The main goal of this research work is to make binarization faster for recognition of a large number of degraded document images on GPU. In this paper, parallel implementation is focused on the well known Niblack's binarization approach for Optical Character Recognition (OCR) systems, since it is one of the most fundamental and important problems in the field of computer vision and pattern recognition. Our work employs extensive usage of highly multithreaded architecture of multi-cored GPU. An efficient use of shared memory is required to optimize parallel reduction in Compute Unified Device Architecture (CUDA). Experimental results show that parallel implementation achieved an average speedup of 20.84x over the serial implementation when running on a GPU named GeForce 9500 GT having 32 cores. Niblack's method of binarization is also evaluated using PSNR, F-measure, NRM, and IND evaluation measures.

General Terms
Document Analysis and Recognition, GPU, Parallel Computing.

Keywords
Binarization; CUDA; OCR; GPU; Parallelization.

1. INTRODUCTION
Graphical Processing Units (GPUs) have been proved its importance in terms of performance as hardware for computer graphics [1]. Many researchers have already been applied GPUs to implement many algorithms in various areas such as image processing, computational geometry, and scientific computation, as well as computer graphics [2-7]. GPUs play important role to speedup processing of document image analysis algorithms because it has more inbuilt execution cores. The parallel implementation of image analysis algorithms using GPU encounters two problems. First, the programmer should master of the fundamentals of GPU and CUDA [8]. CUDA platform is used to implement the parallel implementation of algorithms. Second, in a job it needs much process cooperation between CPU and GPU.

Presented approach of parallelization is based on the first most important phase of character recognition systems named binarization. Binarization is used to produce regions of uniformity within the given image based on some threshold criterion. There are two types of binarization; local and global. Global thresholding methods [9-14] determine a single threshold value for whole image, while local thresholding methods [15-21] are based on the local information for the calculation of threshold value for each pixel. Local thresholding methods are used to remove degradation from poor quality as well as uniform illuminated document images. In old documents degradations always appears due to the environment conditions and may occur due to several reasons of acquisition source type.

Parallel implementations on GPUs have been applied to various numerical problems [22-25] to reduce the computation time without sacrificing the degree of accuracy. Handwritten character recognition is one of the important problems in the field of computer vision. The complexity of the procedure and the high computation cost are the main drawbacks of implementations of fast OCR systems. Computational cost reduction approaches to handwritten character recognition were proposed in [26-30]. Oh at al. implemented neural networks on GPU, which is one of popular algorithm of pattern recognition, and the GPU was used to implement the matrix multiplication of a neural network to enhance the time performance [31]. Jung [32] proposed a Neural Network based text localization in color images. Recently, Singh et al. proposed parallel implementation of well known profiling based segmentation algorithm for Devanagari character recognition on GPU [33].

In the following sections, we present a detailed description of the proposed methodology as well as experimental results that demonstrate the efficiency of the proposed methodology.

2. INTRODCTION TO NVIDIA CUDA ARCHITECTURE
CUDA™ is a general purpose parallel computing architecture introduced by NVIDIA. It contains the CUDA Instruction Set Architecture (ISA) and parallel compute engine in the GPU. The CUDA architecture is programmed using C language, which can then be run with great performance on a CUDA enabled

processor [34]. CUDA-enabled GPUs have hundreds of cores that can collectively run thousands of computing threads. Each core has shared resources, including registers and memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus [35]. Thread hierarchy, shared memories and barrier synchronization are the three key abstractions of CUDA. A kernel can be executed by a one dimensional or two dimensional grids of multiple equally-shaped thread blocks. A thread block is a 3, 2 or 1-dimensional group of threads. Threads within a block can cooperate among themselves by sharing data through some shared memory and synchronizing their execution to coordinate memory accesses. Threads in different blocks cannot cooperate and each block can execute in any order relative to other blocks. The number of threads per block is therefore restricted by the limited memory resources of a processor core.

CUDA kernel function is a fundamental building block of CUDA programs. When launching a CUDA kernel function, a developer specifies how many copies of it to run. We call each of these copies a task. Because of the hardware support of the GPU, each of these tasks can be small, and the developer can queue hundreds of thousands of them for execution at once. These tasks are organized in a two-level hierarchy, block and grid. Small sets of tightly coupled tasks are grouped into blocks. In a given execution of a CUDA kernel function, all blocks contain the same number of tasks. The tasks in a block run concurrently and can easily communicate with each other, which enables useful optimizations such as those of the section “Shared Memory”. GPU’s hardware keeps multiple blocks in flight at once, with no guarantees about their relative execution order. As a result, synchronization between blocks is difficult. The set of all blocks run during the execution of a CUDA kernel function is called a grid.

3. NIBLACK’S BINARIZATION APPROACH

Niblack’s algorithm is a local thresholding algorithm [16]. The threshold value for a pixel is decided by local mean and local standard deviation over a specific window size around each pixel. The local threshold T(x,y) for pixel (x,y) is calculated by formula:

$$T(x,y) = m(x,y) + k \cdot s(x,y) \qquad (1)$$

Where m(x,y) and s(x,y) are the local mean and local standard deviation of the pixels within the local window region. The local window is rectangular in nature and pixel (x,y) is the centre of it. The value of ‘k’ controls the amount of text region inside the local window. To conserve local details and handle local illumination level one requires small window size but if choose too small window size then it will not cover object and eliminate noise in the gray image. Window size of 15x15 and k=-0.2 was recommended by Trier and Jain [36].

4. EVALUATION MEASURES

The method of Niblack is evaluated using the four evaluation measures: F-measure, Peak Signal to Noise Ratio (PSNR), Negative Rate Metric (NRM), and Information to Noise Difference (IND).

- F-Measure

$$F-Measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \qquad (2)$$

Where $Recall = \frac{TP}{TP+FN}$ and $Precision = \frac{TP}{TP+FP}$

- PSNR

$$PSNR = 10 \cdot \log\left(\frac{C^2}{MSE}\right) \qquad (3)$$

Where $MSE = \frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left(I(x,y) - I'(x,y)\right)^2}{MN}$, *I* is the original image and *I’* is binarized.

- NRM

$$NRM = \frac{NR_{FN} + NR_{FP}}{2} \qquad (4)$$

Where $NR_{FN} = \frac{FN}{FN+TP}$ and $NR_{FP} = \frac{FP}{FP+TN}$

- Information to Noise Difference (IND)

We have designed a method to test the quality of the binarized image based on information and noise.

$$IND = Ivalue - Nvalue \qquad (5)$$

Where $Ivalue = \frac{TP}{NB_{GT}}$ and $Nvalue = \frac{FP}{NB_{BI}}$; NB_{GT} and NB_{BI} are the number of black pixels in ground truth and in binarized image respectively. Here *Ivalue* signifies the information preserved in the binarized image and *Nvalue* represents the noise in the binarized image. The value of *IND* ranges between -1 to +1 where +1 means binarized image is the exact copy of ground truth while -1 signifies that binarized image is the invert of ground truth.

5. IMPLEMENTATION

In this work, the implementation of proposed approach is based on the two set of experiments. In the first set of experiment, proposed algorithm is implemented in C language and in second set, parallel implementation is done using CUDA. The following section dictates the detailed description of the parallel implementation of the algorithm.

5.1 Parallel Implementation

In CUDA, it is assumed that both host and device maintain their own DRAM. Host memory is allocated using malloc and device memory is allocated using cudaMalloc. CUDA threads are assigned a unique thread ID that identifies its location within the thread, block and grid. This provides a natural way to invoke computation across the image, by using the thread IDs for addressing. The parallel implementation of algorithm of binarization is shown in the form of pseudo code shown in algorithm 1.

Algorithm 1: Parallel Implementation of binarization algorithm

```
Texture grayImage;
Kernal( windowSize, outputImage){
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int sum=0, sqr_sum=0;
    for( i=y-windowSize to y+windowSize)
    for( j=x-windowSize to x+windowSize)
    {
    int v = grayImage.getPixel( j, i);
    sum += v;
    sqr_sum += v*v;
                }
    Calculate mean & varience;
    Threshold = mean+k*varience;
    If
    (grayImage.getPixel( x, y) <= threshold)
    outputImage.setPixel( x, y) = BLACK;
    else
    outputImage.setPixel( x, y) = WHITE;
                }
Main(){
    dim3 dBlock( BLOCKSIZE, BLOCKSIZE);
    dim3 dGrid( (width+dBlock.x-1)/dBlock.x,
(height+dBlock.y-1)/dBlock.y );
    kernel<<< dGrid, dBlock>>>( windowSize,
outputImage);
}
```

6. HARDWARE SPECIFICATIONS

All the experiments are carried out using the hardware specifications of GPU: GeForce 9500 GT, 1 MB DDR2, No of Processors = 4, No of core =32, RAM 1 GB, Frequency 1.35 GHz, DDR2 and CPU: Intel Core 2 Duo, 2.66 GHZ, No of cores available =2, No of thread=1, No of thread/core=1, Physical Memory =2 GB, DDR2

7. RESULTS AND DISCUSSIONS

For the testing of Niblack's approach of local binarization, we collected a data set of handwritten as well as printed documents from newspapers, old books and from different writers. The collected documents are scanned using a scanner at 300 dpi and tested on the computer specifications shown in content 5. The results of Niblack's binarization approach are shown in fig. 2 that demonstrates the efficiency of this approach. On the basis of visual observation, Niblack's method of binarization recovers text from degraded image but it produce background noise. To make faster the method, we parallelized it on CUDA and achieved an average speedup of 20.84x over the serial implementation when running on a GPU. The comparison of serial implementation over parallel is shown in table 1. Table 1 also shows that execution time depends on the window size and image size in megapixels.

Table 1: Execution time serial over parallel implementation

Window Size	Megapixels	Serial	Parallel	Speed-Up	Speed-Up Average
7	1	0.328	0.013831	23.7148435	22.97599222
	2	0.625	0.027624	22.6252534	
	4	1.25	0.05525	22.6244344	
	8	2.535	0.110483	22.9447064	
	16	5.078	0.221064	22.9707234	
11	1	0.75	0.035703	21.0066381	21.04631935
	2	1.507	0.071432	21.0969873	
	4	3.002	0.142813	21.0204953	
	8	6.016	0.285665	21.0596328	
	16	12.03	0.571555	21.0478432	
15	1	1.39	0.068424	20.3145095	20.34245156
	2	2.797	0.137383	20.359142	
	4	5.594	0.274672	20.3661094	
	8	11.172	0.549496	20.3313582	
	16	22.359	1.099201	20.3411387	
19	1	2.25	0.112249	20.044722	20.04419945
	2	4.507	0.224998	20.0312892	
	4	9.015	0.450076	20.0299505	
	8	18.047	0.899823	20.0561666	
	16	36.125	1.800949	20.058869	
23	1	3.296	0.167096	19.7251879	19.79943118
	2	6.625	0.335233	19.7623742	
	4	13.265	0.669849	19.8029705	
	8	26.572	1.339179	19.8420077	
	16	53.265	2.681401	19.8646155	
		Average Speed-Up			20.8416787

Further, the performance of method is evaluated using F-measure, PSNR, NRM, and IND measures, which show the effectiveness of method shown in table 2. Fig.1 shows the graph of execution time of GPU in seconds vs. window size. Fig.2 shows the graph of speedup vs. window size. Fig. 3 shows the graph of execution time of CPU in seconds vs. window size. Output images of Niblack approach is shown in fig. 4.

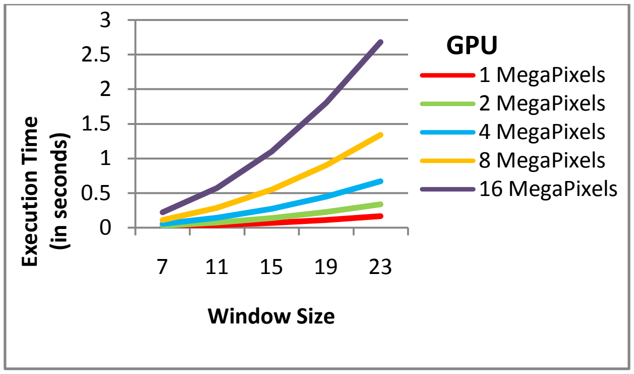


Fig. 1: Execution time of GPU in seconds vs. window size

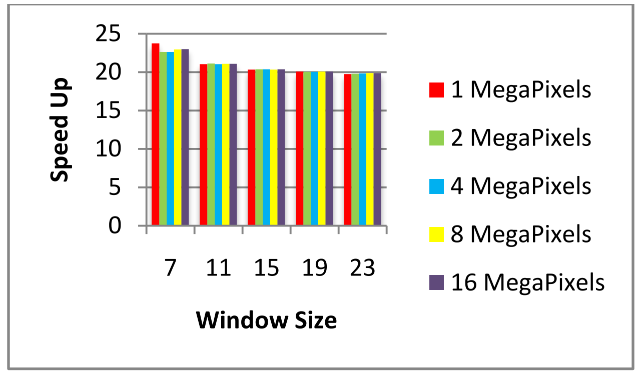


Fig. 2: Speedup vs. Window size

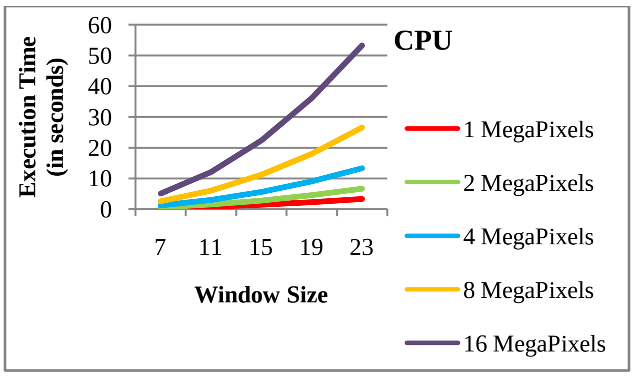
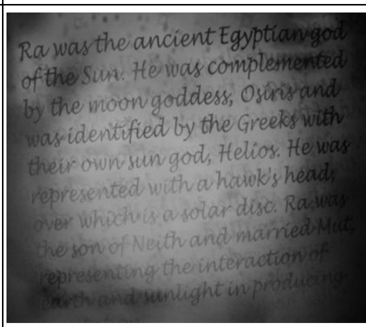
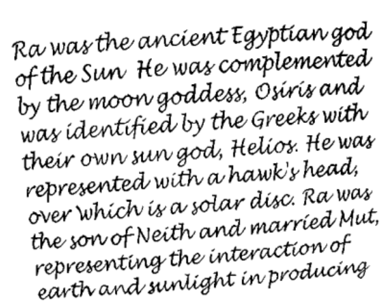
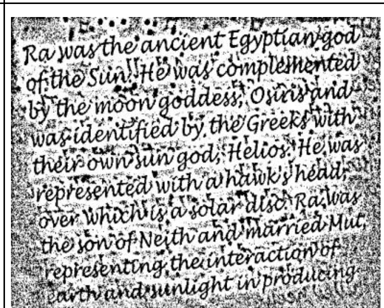
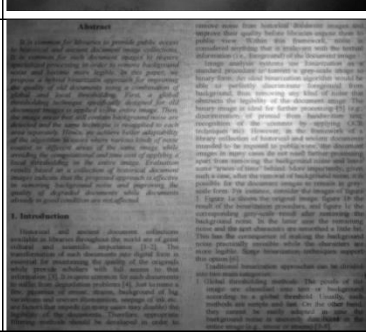
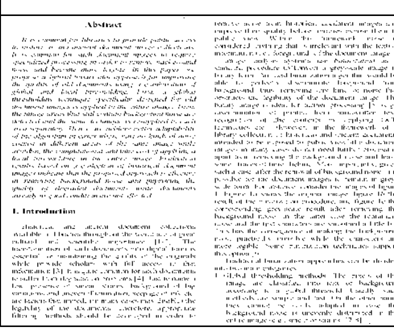


Fig 3: Execution time of CPU in seconds vs. window size

Table 2: Evaluation Measures

Image	F- measure (%)	PSNR (db)	NRM (10-2)	IND
1	56.623	16.319	12.024	0.379
2	63.762	21.012	6.853	0.468
3	49.131	16.867	10.166	0.326
4	62.357	20.410	7.281	0.453
5	49.473	15.831	11.436	0.328

Image No.	Degraded Image	Ground Truth Image	Niblack's Output
1.		 Ra was the ancient Egyptian god of the Sun He was complemented by the moon goddess, Osiris and was identified by the Greeks with their own sun god, Helios. He was represented with a hawk's head, over which is a solar disc. Ra was the son of Neith and married Mut, representing the interaction of earth and sunlight in producing	
2.			

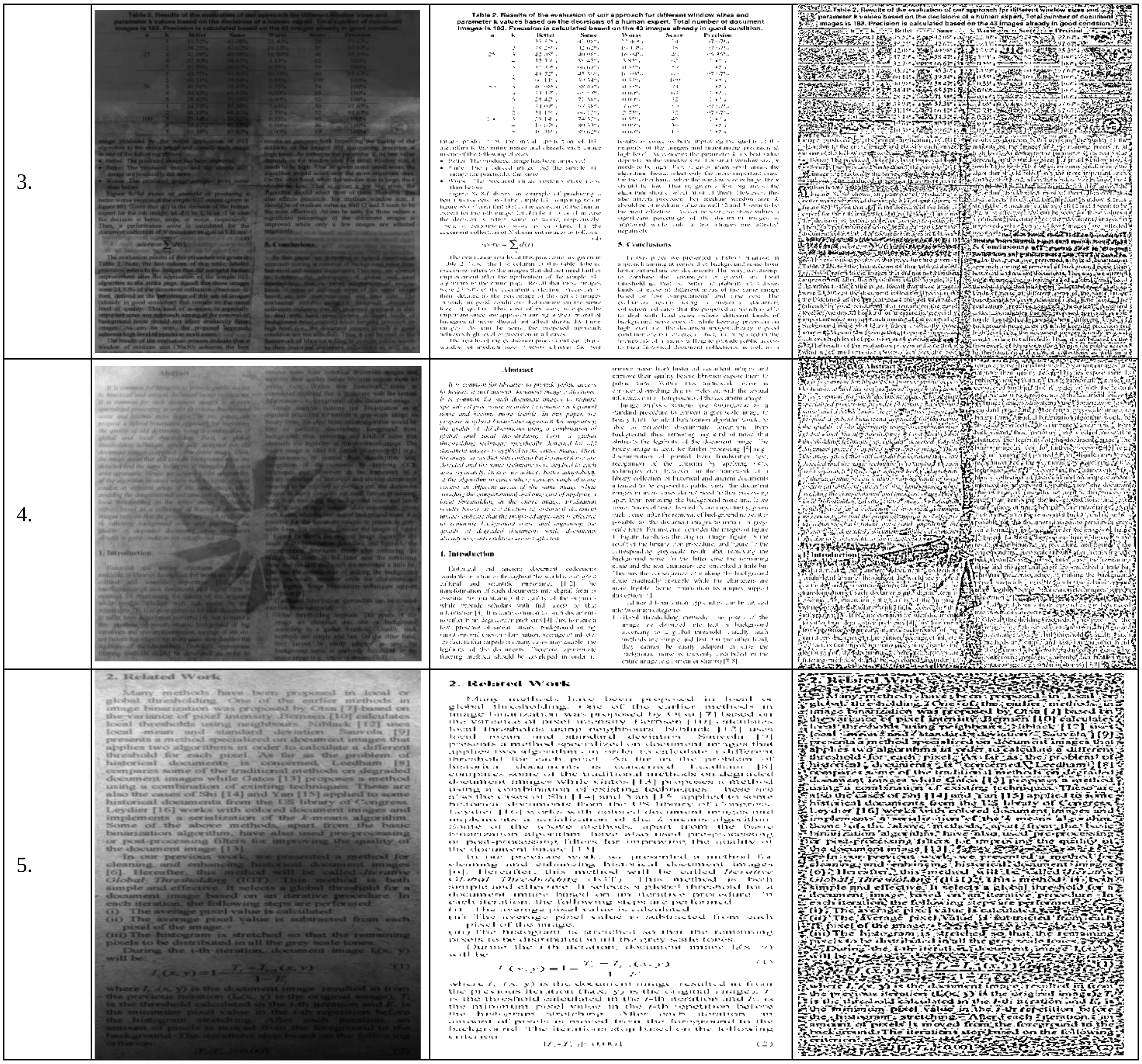


Fig. 4: Output images of Niblack's approach of binarization

8. CONCLUSION

In this research work, a well known Niblack's binarization algorithm has been parallelized and analyzed with evaluation measures. The method is evaluated using PSNR, F-measure, NRM, and IND evaluation measures. The implementation of binarization algorithm on the graphics device is promising with large two dimensional degraded document images. However, Niblack's binarization method produces more background noise but it completely recovers text from severely degraded documents. Adding few post-processing steps make it very attractive for restoration of information from the degraded document images.

CUDA itself has been shown to be an excellent framework to accelerate computational problems in image processing, numerical solving techniques and pattern recognition areas.

9. REFERENCES

[1] Fernando, R and Kilgard, M. J. 2003. The Cg tutorial the definitive guide to programmable real-time graphics. Addison-Wesley.

[2] Moravanszky, A. 2003. Linear algebra on the GPU, in: W.F. Engel (Ed.), Shader X 2, Wordware Publishing, Texas.

[3] Manocha, D. 2003. Interactive geometric & scientific computations using graphics hardware, SIGGRAPH 2003 Tutorial Course #11.

[4] Moreland, K. and Angel E. 2003. The FFT on a GPU. In Proceedings of SIGGRAPH Conference on Graphics Hardware, 112-119.

[5] Mairal, J., Keriven, R. and Chariot, A. 2006. Fast and efficient dense variational Stereo on GPU. In Proceedings of International Symposium on 3D Data Processing, Visualization, and Transmission, 97-704.

[6] Yang, R. and Welch, G. 2002. Fast image segmentation and smoothing using commodity graphics hardware. Journal of Graphics Tools, Vol. 17, (4), 91-100.

[7] Fung, J. and Man, S. 2005. OpenVIDIA: Parallel GPU computer vision. In Proceedings of ACM International Conference on Multimedia, 849-852.

[8] Jang, H., Park, A. and Jung, K. 2008. Neural network implementation using CUDA and OpenMP. In Proceeding of Computing: Techniques and Applications, (DICTA), IEEE, 155 – 161.

[9] Otsu, N. 1979. A threshold selection method from gray level histograms. IEEE Trans. on Systems, Man and Cybernetics, Vol. 9, 62-66.

[10] Yu, B., Jain, A. and Mohiuddin, M. 1997. Address block location on complex mail Pieces," In Proceeding of International Conference of Document Analysis and Recognition, IEEE, 897-901.

[11] Rosenfeld, A. and Kak, A.C. 1982. Digital picture processing, second ed., Academic Press, New York.

[12] Kittler J. and Illingworth J. 1985. On threshold selection using clustering criteria. IEEE Trans. Systems Man Cybernetics, Vol. 15, 652–655.

[13] Brink, A.D. 1992. Thresholding of digital images using two-dimensional entropies. Pattern Recognition, Vol. 25, 803–808.

[14] Yan, H. 1996. Unified formulation of a class of image thresholding techniques. Pattern Recognition, Vol. 29, 2025–2032.

[15] Bernsen, J. 1986. Dynamic thresholding of grey-level images. In Proceeding of International Conference of Pattern Recognition, 1251-1255.

[16] Niblack, W. 1986. An Introduction to digital image processing, Prentice-Hall, Englewood Cliffs, NJ, 115–116.

[17] Sauvola, J. and Pietikainen, M. 2000. Adaptive document image binarization. Pattern Recognition, Vol. 33, 225–236.

[18] Kim, I.K., Jung, D.W. and Park, R.H. 2002. Document image binarization based on topographic analysis using a water flow model. Pattern Recognition, Vol. 35, 265–277.

[19] Gatos, B., Pratikakis, I. and Perantonis, S. J. 2006. Adaptive degraded document image binarization. Pattern Recognition, Vol. 39, 317–327.

[20] Chang, Y.F., Pai, Y.T. and Ruan, S.J. 2008. An efficient thresholding algorithm for degraded document images based on intelligent block detection. In Proceeding of IEEE International Conference on Systems, Man, and Cybernetics, 667-672.

[21] Valizadeh, M., Komeili, M., Armanfard, N. and Kabir, E. 2009. Degraded document image binarization based on combination of two complementary algorithms. In Proceeding of International Conference of Advances in Computational Tools for Engineering Applications, IEEE, 595-599.

[22] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A. E. and Purcell, T. J. 2005. A survey of general-purpose computation on graphics hardware. In proceeding of Eurographics, State of the Art Reports, 21–51.

[23] Larsen, E. S., McAllister, D. 2001. Fast Matrix Multiplies using Graphics Hardware. In Proceeding of International Conference for High Performance Computing and Communications, 159-168.

[24] Trendall C. and Stewart, A. J. 2000. General calculations using graphics hardware with applications to interactive caustics. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, 287-298.

[25] Li, Wei, Wei, Xiaoming, A. and Kaufman, 2001. Implementing lattice boltzmann computation on graphics hardware. In proceeding of the International Conference for High Performance Computing and Communications.

[26] Mizukami, Y., Koga, K. and Torioka, T. 1994. A handwritten character recognition system using hierarchical extraction of displacement. IEICE, J77-D-II(12):2390–2393.

[27] Kruger, J. and Westermann, R. 2003. Linear operators for GPU implementation of numerical algorithms. In Proceedings of SIGGRAPH, San Diego, 908- 916.

[28] Steinkraus, D., Buck, I., and Simard, P. Y. 2005. GPUs for machine learning algorithms. In proceeding of International Conference of Document Analysis and Recognition, 1115-1120.

[29] Mizukami, Y. and Koga, K. 1996. A handwritten character recognition system using hierarchical displacement extraction algorithm. In Proceeding of International Conference of Pattern Recognition, volume 3,160–164.

[30] Ilie, A. Optical character recognition on graphics hardware. Downloaded from www.cs.unc.edu/~adyilie/IP/Final.pdf

[31] Oh, K.S. and Jung, K. 2004. GPU implementation of neural networks. Pattern Recognition, Elsevier, 1311-1314.

[32] Jung, K. 2001. Neural Network-based text localization in color images. Pattern Recognition Letters, Vol. 22, (4), 1503- 1515.

[33] Singh, B.M., Mittal A., and Ghosh, D. 2011. Parallel implementation of Devanagari text line and word segmentation approach on GPU. International Journal of Computer Applications 24(9):7–14.

[34] NVIDIA CUDA Programming Guide Version 2.0, available at www.nvidia.com/object/cuda_develop.html.

[35] NVIDIA Corporation: NVIDIA CUDA programming guide. Jan 2007, available at http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

[36] Trier, O.D. and Jain, A.K. 1995. Goal-directed evaluation of thresholding methods, IEEE Trans. Pattern Anal. Mach. Intell. 17 (12) 1191–1201.