GPU Implementation of Spiking Neural Networks for Color Image Segmentation

Ermai Xie, Martin McGinnity, QingXiang Wu
Intelligent Systems Research Center
University of Ulster at Magee
Londonderry, BT48 7JL, Northern Ireland, UK
{q.wu, tm.mcginnity, ermai}@ulster.ac.uk

Jianyong Cai, Rontai Cai
School of Physics and OptoElectronics Technology
Fujian Normal University
Fuzhou, 350007, China
{cjy, rtcai}@fjnu.edu.cn

***Abstract*—Spiking neural networks (SNN) are powerful computational model inspired by the human neural system for engineers and neuroscientists to simulate intelligent computation of the brain. Inspired by the visual system, various spiking neural network models have been used to process visual images. However, it is time-consuming to simulate a large scale of spiking neurons in the networks using CPU programming. Spiking neural networks inherit intrinsically parallel mechanism from biological system. A massively parallel implementation technology is required to simulate them. To address this issue, modern Graphic Processing Units (GPUs), which have parallel array of streaming multiprocessors, allow many thousands of lightweight threads to be run, is proposed and proved as a pertinent solution. This paper presents an approach for implementation of an SNN model which performs color image segmentation on GPU. This approach is then compared with an equivalent implementation on an Intel Xeon CPU. The results show that the GPU approach was found to provide a 31 times faster than the CPU implementation.**

Keywords- graphic processing units; spiking neural network; computer unified device architecture; colore image segmentation.

I. INTRODUCTION

The human visual system is most powerful image processing system, and is hugely complex and massively parallel processors which have about 130 million neurons as photoreceptors absorbing light, yet roughly 1.2 million axons of ganglion cells transmitting information from the retina to the brain. To simulate behaviours of the human visual system, implementation of spiking neural networks (SNN) become an important computational technology. Currently different models of neurons in spiking neural networks have been proposed, such as Hodgkin-Huxley model [1], the conductance based integrate and fire [2], and a simple spiking neuron model (lzhikevich model) [3]. The dynamic properties have been described in [4]. These models are usually contains a group of deferential equations. It is time-consuming to simulate a large scale of spiking neural network using Traditional Central Processing Unit (CPUs). Traditional CPUs process images in a sequential fashion and cause the time cost to be significant [5]. GPU implementation has provided a new solution for this problem [10, 11]. Therefore, in this paper, a GPU-based implementation approach is proposed to simulate a large scale of spiking neural network. The intrinsically parallel mechanism in biological system is mapped to the parallel mechanism in GPU. Since there are the differences between the two parallel mechanisms, a strategy of transform from neural networks to CUDA architecture is required. In this paper, a SNN model for color image segmentation [6] is implemented using the parallel mechanism of GPU. In the implementation, the strategy and techniques for implementation of large scale of SNNs are addressed.

The remainder of this paper is organized as follows. In Section II, GPU and the Compute Unified Device Architecture (CUDA), which is the approach to programming NVDIA GPU, are introduced. The architecture of the neural network for segmentation of color images is presented in Section III. In Section IV, GPU-based implementation of the SNN model is presented. Experimental results, which obtained from implementation of the SNN on the CUDA hardware and a CPU implementation, are shown in Section V. Section VI will outline the direction of further research which will use GPU based CUDA programming to explore large scale SNNs for different bio-inspired neuron models.

II. GPU AND CUDA

Graphics Processing Unit (GPU) is a hardware unit traditionally used in computer to render graphic information for users. Unlike CPU is designed to control flow and store data caching, GPU is design for compute-intensive, consists of massive multi-threading cores that operate together and devotes more transistors to data processing in parallel. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. Because of the strong ability for parallel computing, the GPUs have been regarded as a way to bring huge amounts of processing power to computers and attract research communities.

CUDA, which is a parallel computing engine in GPUs developed by NVIDA, allows programmer get access to the virtual instruction set, memory of the parallel computation like CPUs and run lightweight threads. CUDA computing engine in NVIDIA graphics processing units (GPUs) is accessible to software developers through variants of industry standard programming languages. Programmers can use 'C for CUDA' (C with NVIDIA extensions and certain restrictions) to code algorithms for execution on the GPU.

978-1-4244-9305-0/11/$26.00 ©2011 IEEE

Therefore, the parallel architecture can be used to implement large scale of spiking neural networks. Since the GPUs of the most powerful class typically interface with the PC motherboard by means of an expansion slot such as PCI Express (PCIe) or Accelerated Graphics Port (AGP) and can usually be replaced or upgraded with relative ease, this makes that GPU-based spiking neural networks are possible to be applied to industry products.

III. SNN Model Inspired By The Vsual System For Color Image Segmentation

In order to demonstrate GPU-based implementation of SNNs, an SNN model for color image segmentation proposed by Q. Wu et al is selected. The principle of this model is based on biological evidence which shows that the retina contains two forms of photosensitive neurons: rods and cones [7]. Rod cells are highly sensitive to light and respond in dim light and dark conditions. Cones consist of three sub-types have primary sensitivity of red, green, and blue lights. Inspired by this structure of retina, this SNN model takes separate color images into different colors and gray scale as input features. Let the input image dimension be $P \times Q$. A color pixel in the image is labeled as $P_{x,y}$, where $x = 1, 2, ..., P$ and $y = 1, 2, ..., Q$. Since biological cones have small receptive fields and have high spatial resolution, $RC_{x,y}$ is designed to represent this structure for three types of cones corresponding to three colors, and labeled as R, G, and B. The large receptive field $RN_{x,y}$ is designed to simulate rod which respond to gray scale of an image. Each of receptive field forms two ON/OFF pathways. Therefore, there are eight different features is extracted and then used for segmentation. An error back-propagation (BP) neural network is selected as a classifier to segment the visual image. The output layer is the segmentation result matrix with the same dimension as the input image. The architecture of this SNN model is shown as Fig. 1.

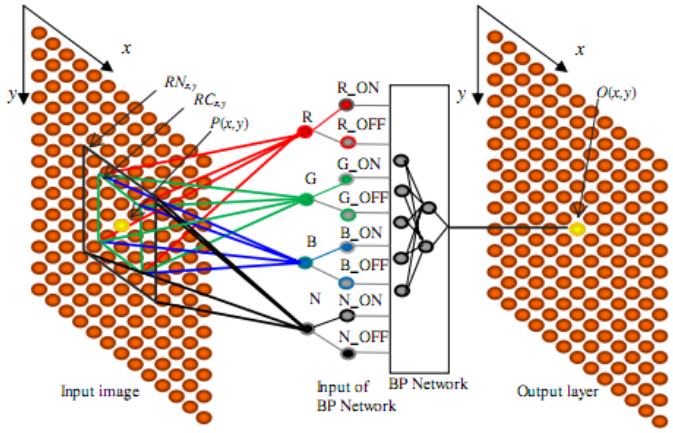


Fig. 1. SNN model for segmentation of color image

In this model, a simplified conductance-based integrate-and-fire neuron model is used to simulate neurons in the network. First of all, normalization is necessary for the strength of input lights. Suppose that $P_{x,y}(t)$ represents the strength of red light from the input image pixel at a point (x, y) at time t. Let $R_{x,y}(t)$ represent the normalized strength:

$$R_{x,y}(t) = P_{x,y}(t) / P_{\max} \tag{1}$$

Suppose that each red photon receptor transfers the red-light strength to a synapse current $I_{R\_ON(x,y)}$, which can be represented as follows:

$$I_{R\_ON(x,y)} = \alpha R_{x,y}(t) \tag{2}$$

where α is a constant for transformation from the red-light strength to current. For the OFF pathway, the synapse current can be represented as follows:

$$I_{R\_OFF(x,y)} = \beta(1 - R_{x,y}(t)) \tag{3}$$

where β is a constant for transformation for the OFF pathway. According to the integrate-and-fire neuron model [6, 8, 9], the potential of an ON neuron $R\_ON(x, y)$ is governed by the following equations:

$$A_1 = g_l(E_l - v_{R\_ON(x,y)}(t)) \tag{4}$$

$$A_2 = \sum_{(x',y') \in RC_{x,y}} w_{x',y'} I_{R\_ON(x',y')}(t) \tag{5}$$

$$A_3 = \sum_{(x^*,y^*) \in RF^*_{x,y}} w_{lateral} S_{R\_ON(x^*,y^*)}(t) \tag{6}$$

$$c\frac{dv_{R\_ON(x,y)}(t)}{dt} = A_1 + A_2 + A_3 + I_0 \tag{7}$$

where g_l is the membrane conductance, E_l is the reverse potential, $v_{R\_on(x,y)}(t)$ is the membrane potential of the neuron $R\_ON(x, y)$, c represents the capacitance of the membrane, I_0 is simulated by an average current produce by background noise, $w_{lateral}$ is the synapse strength of lateral connections from the neighborhood neurons, $RF^*_{x,y}$ is a set of neighborhood neurons, and $S_{R\_ON(x^*,y^*)}(t)$ is a spike train from neighborhood neurons. $w_{x',y'}$ is used to simulate the centre-ON and surround-OFF receptive field, which is represented by the following expression.

$$w_{x',y'} = w_0 e^{-\frac{(x-x')^2+(y-y')^2}{\delta^2}} - w_1 e^{-\frac{(\sqrt{(x-x')^2+(y-y')^2}-\Delta)^2}{\delta^2}} \tag{8}$$

where w_0 and w_1 are used to determine the maximal value of the weight distribution, δ is a decay constant for the distribution away from the centre point (x, y), Δ is corresponds to the radius of the OFF surround circle. If the membrane potential passes a threshold v_{th}, then the neuron generates a spike. Let $S_{R\_ON(x,y)}(t)$ represent the spike train generated by the neuron such as that:

$$S_{R\_ON(x,y)}(t) = \begin{cases} 1 & \text{if neuron } R\_ON(x,y) \text{ fires at time } t. \\ 0 & \text{if neuron } R\_ON(x,y) \text{ does not fires at time } t. \end{cases} \tag{9}$$

Similar representation is used to $S_{R\_OFF(x,y)}(t)$. The firing rate of neuron $F_{R\_ON(x,y)}$ is calculated by the following expression:

$$F_{R\_ON(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{R\_ON(x,y)}(t) . \quad (10)$$

By analogy, using synapse current (2), a firing rate of *R_OFF* neuron can be obtained.

$$F_{R\_OFF(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{R\_OFF(x,y)}(t) . \quad (11)$$

For color green and blue, we can have

$$F_{G\_ON(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{G\_ON(x,y)}(t) . \quad (12)$$

$$F_{G\_OFF(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{G\_OFF(x,y)}(t) . \quad (13)$$

$$F_{B\_ON(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{B\_ON(x,y)}(t) . \quad (14)$$

$$F_{B\_OFF(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{B\_OFF(x,y)}(t) . \quad (15)$$

By analogy, let $N_{x,y}(t)$ represent the normalized value of the gray scale image at the point (x, y) at time t. For neuron $N\_ON(x,y)$, we have

$$I_{N\_ON(x,y)} = \alpha N_{x,y}(t). \quad (16)$$

$$I_{N\_OFF(x,y)} = \beta(1 - N_{x,y}(t)). \quad (17)$$

These currents inject to neuron $N\_ON(x,y)$. We can have

$$F_{N\_ON(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{N\_ON(x,y)}(t) \quad (18)$$

$$F_{N\_OFF(x,y)}(t) = \frac{1}{T}\sum_{t}^{t+T} S_{N\_OFF(x,y)}(t) \quad (19)$$

Suppose $O(x,y)$ is the output of the BP neural network. Let ($F_{R\_ON(x,y)}$, $F_{R\_OFF(x,y)}$, $F_{G\_ON(x,y)}$, $F_{G\_OFF(x,y)}$, $F_{B\_ON(x,y)}$, $F_{B\_OFF(x,y)}$, $F_{N\_ON(x,y)}$, $F_{N\_OFF(x,y)}$) represent the input vectors of the network. The network is represented by the following expression.

$$O(x,y) = NET(F_{R\_ON(x,y)}, F_{R\_OFF(x,y)}, F_{G\_ON(x,y)}, F_{G\_OFF(x,y)}, F_{B\_ON(x,y)}, F_{B\_OFF(x,y)}, F_{N\_ON(x,y)}, F_{N\_OFF(x,y)})$$

$O(x,y)$ can be defined as a classification label for a segmentation area. For example, this approach is applied to segment leukocytes from a blood smeared image as shown in Fig. 2. Four label numbers {0, 1, 2, 3} are defined. "0" represents background in a smear image. "1" represents erythrocyte. "2" represents cytoplasm of leukocytes. "3" represents nucleus of leukocytes. Therefore, 4 areas can be obtained in the output layer.

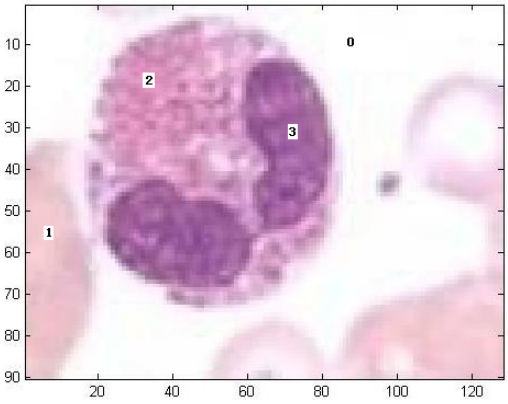


Fig. 2. Blood smeared image

IV. Implementation Based on GPU

The GPU is especially well-suited to address problems that can be expressed as data-parallel computations. Implementation of SNN models is such a problem, where the same neuron model is executed on many data elements in parallel. Because the same neuron model is executed for each data element, there is a lower requirement for sophisticated flow control; and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. Therefore, GPU is suitable to implement SNN models.

The original SNN model [6] has been implemented in Matlab. However, it will take a very long time to segment an image when the size of image is large. Since the network model is fully parallel architecture, which uses an independent process to simulate the behaviors of each neuron corresponding to each pixel in the image, CUDA has potential to accelerate the algorithm. At first the network model has been redesigned using architecture of CUDA. The model in CUDA architecture is implemented in two parts. The first part is a parallel algorithm for extraction of eight ON/OFF features. The second part is a parallel algorithm for segmentation of the image using BP neural network based on the inputs of eight features.

In the first part of the CUDA implantation, independent threads are used to process the task corresponding to each pixel in the image. The formal algorithm for extraction of the eight ON/OFF features is listed as follows.

Formal Algorithm for extraction of eight ON/OFF features

1. Begin
2. Load input image
3. Determine image size
4. Allocate and initialize memory on GPU for image, Receptive fields, Intermediate layer and output layers
5. Copy input image and RF data to the GPU
6. Calculate Gray level for each pixel in parallel
7. Normalize Red, Green, Blue and Gray level for each pixel in parallel according to equation (1)-(3)

8. Calculate A2 for eight different channels of each pixel based on equation (5) in parallel
9. For each time step do following steps in parallel
10. Calculate the membrane potential to decide if neurons fired in parallel according to Equation (4)-(7)
11. Record times of neurons fired according to membrane potential and threshold
12. End for
13. Copy output data from the GPU
14. Free GPU memory
15. Save output image
16. End

This formal algorithm shows the basic implementation flow for the SNN to extract ON/OFF image features. The whole flow consist of two types of programs, the first type is only run once, such as calculate gray level, do normalization and calculate value of A2, i.e. integration of image intensity in the receptive field. As the images are static in this case, i.e. the value $I_{R\_ON(x,y)}$, and $I_{R\_OFF(x,y)}$ are not changing with time. The second type of program is executed at every time step to simulate the dynamic behaviors of spiking neurons.

In the second part of the CUDA program, a BP neural network is implemented using a thread for each neuron in the output layer. The BP neural network is used to classify pixels according to the eight ON/OFF features. Each pixel in the output layer is indicated using labels '0', '1', '2', and '3' for the input blood smeared image. The formal algorithm based on CUDA is as follows.

Formal Algorithm for segmentation of image using BP network.

1. Begin
2. Load input image
3. Determine image size
4. Allocate and initialize memory on GPU for image, weights of neural networks, features from 8 channels and output layers
5. Copy input image and weights of NN, features to the GPU
6. Run BP network for each pixel with its own weights and features to segment the pixel in parallel
7. Mark the pixel into four different categories
8. Copy output data from the GPU
9. Free GPU memory
10. Save output masks
11. End

As GPU memory latency is about 100 times smaller than global memory, the strategy for using local memory and avoiding non-coalesced global memory access is considered in the implementation of this SNN model.

For comparison, the SNN model has also been implemented using C#, and C.

V. EXPRIEMENTAL RESULTS

In our experiments, the following parameters for the network were used in the experiments. v_{th} = -60 mv. E_l= -70 mv. g_l=1.0 μs/mm^2. c=8 nF/mm^2. τ = 4ms. T=400ms. These parameters can be adjusted to get a good quality output image. α=4.8. β=11.2. I_0=7μA. $w_{Lateral}$=0.01. w_0=1. w_1=1/3. Δ=1.5. δ=1.5. These parameters can be determined by trial/error method according to single neuron behaviors. A blood smeared image, shown in Fig. 2, presents to the network. The eight firing rate maps are obtained from eight ON/OFF pathways which are shown in Fig. 3.

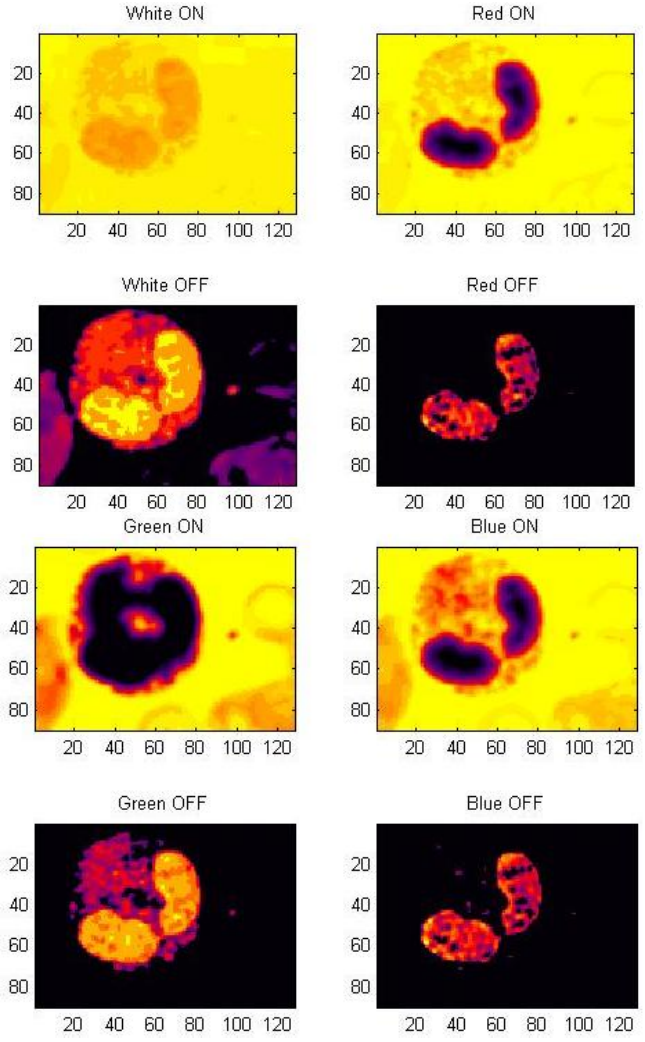
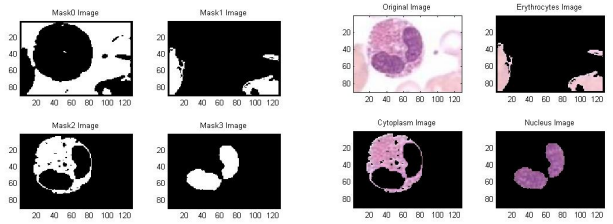


Fig. 3. Eight firing rate maps

The eight values of each pixel are regarded as input for the BP neural network. The BP neural network is trained using the sample points from 4 different areas, i.e. background, erythrocyte, cytoplasm, and nucleus of leukocytes. The trained network is used to segment the blood smeared image. The results are shown in Fig.4(a). Mask0, Mask1, Mask2 and Mask3 represent background, erythrocyte, cytoplasm, and nucleus respectively. Fig.4(b) is shown the segmentation areas filtered using these masks. More results are shown in Fig. 5.



(a)Masks (b)Segmentation areas

Fig .4. Segmentation masks and areas

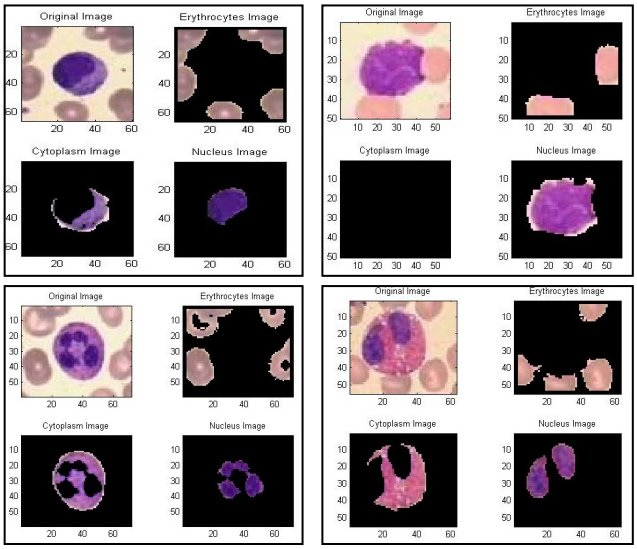


Fig .5. Segmentation results for blood smeared images

Table 1. Comparison results of run time

Platform / Image	Matlab	.NET (C#)	C	CUDA (GPU)
3072 * 4080	-	-	>20 hours	2294 s
1536 * 2040	>46.2 hours	5150 s	1852 s	59.07s
482 * 640	>4.2 hours	472 s	169 s	5.899 s
90*128	549 s	17 s	6 s	0.238 s

In order to compare the running time, different size images are used to test the algorithms implemented in Matlab, C#, C and CUDA respectively. Comparable results are shown in Table 1. This results are achieved using a PC system with Intel(R) Xeon(R) CPU X5550 and NVIDIA Quadro FX 3800. It can be seen that CUDA implement is 31 times faster than C for an image with size 1536X2040. It is incredible faster than Matlab.

VI. Conclusion and Further Work

This paper presents a general implementation of spiking neural networks using CUDA architecture in the GPU. The algorithm of spiking neural network model for color image segmentation is used to demonstrate the implementation techniques. The experimental results show that CUDA implementation of spiking neural networks can speed up running time over 31 times for images with size 1536X2040. In this study, only multiple threads in CUDA have been used, it is promising to speed up further if texture mechanism of CUDA is applied. The further work is to develop a theory to bridge two architectures of spiking neural networks and CUDA, and apply this implementation approach to more complicated spiking neuron models.

VII. Refernce

[1] W.Gerstner, W. Kistler, “Spiking Neuron Models: Single Neurons”, populations, Plasticity. Cambridge University Press, 2002.

[2] L. P. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, , A. Belatreche, and J. Harkin, “Challenges for large-scale implementations of spiking neural networks on FPGAs”, Neurocomputing, vol. 71, pp.13–29, 2007.

[3] E. M. lzhikevich, “Simple Model of Spiking Neurons”, IEEE Trans on Neural Networks, vol.14, no.6, pp.1569-. 1572, 2003.

[4] E.M. Izhikevich, “Which model to use for cortical spiking neurons?” IEEE Transactions on Neural Networks, Vol. 15, No. 5, pp.1063 – 1070, Sept. 2004

[5] M. Khan, D. Lester, L.Plana , A. Rast, X. Jin, E. Painkras, and S. Furber, “SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor”, In Proc. of Intl Joint Conf. on Neural Networks (IJCNN2008), pp. 2849–2856.

[6] Q. X. Wu, T. M. McGinnity, Liam Maguire, G.D. Valderrama-Gonzalez, and P. Dempster, “Colour Image Segmentation Based on a Spiking Neural Network Model Inspired by the Visual System”, Springer, LNCS 6215, pp. 49–57.

[7] E. R. Kandel, J. H Shwartz, Principles of neural science. Edward Amold (Publishers) Ltd., 1981.

[8] Q.X. Wu, T.M. McGinnity, L.P. Maguire, A.Belatreche and B. Glackin: “2D Co-ordinate Transformation Based on a Spike Timing-Dependent Plasticity Learning Mechanism”, Neural Networks, vol. 21, pp.1318-1327, 2008.

[9] Q.X. Wu, T.M McGinnity, L. P. Maguire, A.Belatreche and B. Glackin, “Processing Visual Stimuli Using Hierarchical Spiking Neural Networks”, Neurocomputing, Vol.71, No.10-12, pp.2055-2068, 2008.

[10] J. M. Nageswaran, et al, “A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors”, Neural Networks, Vol. 22, No. 5-6, July, 2009.

[11] F. Bernhard and R. Keriven, "Spiking Neurons on GPUs", Lecture Notes in Computer Science, 2006, Vol. 3994, pp. 236-243,2006.