IJICTR International Journal of Information & Communication Technology Research

ITRC

Volume 5- Number 3- Summer 2013

Accelarated Optical Character Recognition on Graphics Processing Units

Ehsan Arianyan
Electrical Engineering Department
Amirkabir University of Technology
Tehran, Iran
ehsan_arianyan@aut.ac.ir

S. Ahmad Motamedi
Electrical Engineering Department
Amirkabir University of Technology
Tehran, Iran
motamedi@aut.ac.ir

Iman Arianyan
Electrical Engineering Department
Amirkabir University of Technology
Tehran, Iran
iman_aryanian@aut.ac.ir

Mohammad Motamedi
Electrical Engineering Department
Amirkabir University of Technology
Tehran, Iran
mtmd@aut.ac.ir

Received: February 6, 2013- Accepted: May 21, 2013

***Abstract* — Optical Character Recognition (OCR) is a technique by the help of which the optical characters are identified automatically by a computer. There are many methods for OCR, one of which is neural network that we use. Unfortunately, the long training and testing time of these networks is disturbing, but we healed this problem by mapping our network on graphics card by using Jacket which is the product of Accelereyes group. By so doing, we achieved the speedup of up to twelve factors. Graphics Processing Units (GPUs) have parallel structure containing many cores capable of running thousands of threads in parallel. We train a multi-layer perceptron network using back propagation rule which has a degree of parallelism that is suitable for implementation on new graphics card. We examine the Persian characters that are typed on the new system of Farsi license plates to make a database of characters uses in this system and apply them as train and test data for our network.**

Keywords- Neural network; OCR; GPU; Jacket; CUDA

I. INTRODUCTION

Graphics Processing Unit (GPU) is a highly parallel computing device and its main task was first only graphics rendering. However, the GPU has gradually developed in recent years to become a more general processor, and today is used for both sophisticated graphics rendering and even scientific applications. Generally speaking, the GPU has become a powerful device for the execution of data-parallel, arithmetic intensive applications in which the same operations are carried out on many elements of data in parallel. This kind of computation is called Single Instruction Multiple Data (SIMD) and the GPU only offers speedup for the computations that are in this format. GPUs have advanced at an astonishing rate, currently capable of delivering over 1 TFLOPS of single precision performance and over 300 GFLOPS of double precision while executing up to 240 simultaneous threads in one low-cost package. As such, GPUs have gained significant popularity as powerful tools for high performance computing (HPC) achieving 2-100 times the speed of their x86 counterparts in various applications.

Artificial Neural Network (ANN) is a nonlinear model that is typically like a black box trained to a specific task on a large number of data samples. Slow training of neural networks has always been a great discomfort for scientists. Multi-layer perceptron (MLP) is one of these structures that is too slow to be a faithful solution for some applications. However, there is a high parallelism in both the architecture of MLP network, and its algorithms. Hence, by implementing it on modern new GPUs we can get huge speedup at relatively low cost.

When the time is more than a threshold (for example an hour) this speedup is clearly admired, and that is the case in online police car license plate recognition system. Due to the heavy computation and the speed of the cars pass across the police camera, test time of the network is also important. So, we devised a new system in which we first train two separate MLP neural networks independently with characters and numbers that are used in license plate, and then copy the weights of these two network for as many as networks that are needed so that they can be used independently for recognition of different plates simultaneously. Moreover, seven characters that are present in a Farsi license plate are recognized at once with the parallel many-core GPU structure. The changes made to the algorithm of back propagation to become suitable for a parallel architecture of GPUs will be discussed in later sections.

There are many language bindings that help the programmer, by knowing some extra extensions, easily transform his or her code written in any language to a code that is familiar for GPU and run the output code on GPU. Table I lists these bindings.

TABLE I. LANGUAGE BINDINGS

Language	Binded language
.NET	CUDA.NET
MATLAB	Jacket, GPUmat
Fortran	FORTRAN CUDA, PGI CUDA Fortran Compiler
Perl	KappaCUDA
Ruby	KappaCUDA
Lua	KappaCUDA

A. *Paper Organization*

In the sections that follow, we first review related work of other researchers, and then explain optical character recognition. After that, we talk about MLP and back propagation algorithm for training the neural network, and how we changed the algorithm to be appropriate for implementing on GPUs. In the next section we intoduce CUDA and Jacket, and in the final section we present our experimental results and compare the speedup we get from GPU implementation of a back propagation learning algorithm against its CPU counterpart and conclude at the end.

II. RELATED WORKS

There has been a great effort to improve the algorithms of optical character recognition so that their error lessens to a degree that could be trusted and reliably recognize characters. However, a few works have been done to both make the recognition procedure faster with the help of GPUs, and at the same time make a network which has little error. Daniel and his fellow worker implemented a restricted Boltzmann machine network on GPU and tried to take advantage of parallelism in this kind of neural network to map it on GPU parallel structure. They tested their algorithm on a NVIDIA GTX280 GPU, resulting in a computational speed of 672 million connections-per-second and a speedup of 66 fold over an optimized C++ program running on a 2.83GHz Intel processor. [1]. Jang and his fellow workers implemented a multi-layer perceptron neural network on both CPU and GPU using CUDA and OpenMP. They reported that their implementation times on GPU showed about 15 times faster than implementation using CPU and about 4 times faster than implementation on only GPU without OpenMP [2]. Prabhu took advantage of GPUs for speeding SOM neural network and chose NVIDIA GeForce 6150 Go with Microsoft Research Accelerator as the high level library as the implementation platform. He mentioned that if the computation is not heavy enough or the algorithm is not parallel, the gain of speed that can be achieved is not too much so that it deserves implementation on GPU [3]. Strigl and his fellow worker presented the implementation of a framework for accelerating training and classification of Convolutional Neural Networks (CNNs) on the GPU. They reported that depending on the network topology, training and classification on the GPU performs 2 to 24 times faster than on the CPU [4]. Bernhard and his colleague used GPUs to increase the speed of two image-segmentation algorithms using spiking neural networks and one multi-purpose spiking neural-network simulator. They reported a speed increase between 5 and 20 times faster in all the algorithms implemented [5]. Moorkanikara and his fellow worker presented the acceleration of Address Event Representation (AER) based spike processing using a GPU. Their implementation can achieve a kernel speedup of up to 35x on a single NVIDIA GTX280 board when compared to a CPU-only implementation [6]. While Suresh in “Parallel implementation of back-propagation algorithm in networks of workstations” had adorable efforts on parallelizing the back propagation algorithm on the Network of Workstations (NOWs), the fact that accessing huge parallel processing facilities is not always cheap and even possible is undeniable [14]. Casey in the “A Processor-Based OCR System” also tried to divide the OCR process to two sequential parts. In their work the first division of processes was planned to be handle by an interface and the second part by a separate processor. This separation of the OCR processes in their work can be considered as pipeline especially for sequential inputs [15]. It is worthy to mention that, however speeding up the OCR procedure is important; there are still some other important issues about OCR including the accuracy of recognition and multidimensional OCRs. There are also lots of lots of works on these two areas. As a case in point Roy *et al.* in “Multi-Oriented English Text Line Extraction Using Background and Foreground Information” [16] and Pal *et al.* in “Multioriented and curved text lines extraction from Indian documents“ [17] worked on Recognizing the characters which are not in a regular form.

Obviously there is a direct relation between complexity and accuracy. In other words, more accurate algorithms usually have more complicated procedure. Such complicated procedure needs more process and as a result more process time. Ergo, here there is a tradeoff between the process time and accuracy. Based on aforementioned facts, it is pretty easy to speed up the OCR process by decreasing the accuracy. However, in this paper we are concentrating on parallelizing the OCR process and reducing the execution time without diminishing the accuracy. Note that Beg *et al.* in “Hybrid OCR Techniques for Cursive Script Languages - A Review and Applications” concentrates on this issue too [18].

However, their research is focusing on implementing the OCR process on the hardware. Our research avoids hardware implementation due to this fact that while hardware implementation reduces the execution time, it makes the implemented algorithm inflexible; implementing the algorithms on hardware usually makes their modification arduous, expensive and even impossible.

III. OCR

OCR is abbreviation of Optical Character Recognition and is a technique which is used for automatic recognition of letters that are in a handwritten or printed text by a computer without intervene of human and translate human-readable characters to machine-readable codes. OCR is applicable in many applications such as in automatic license plate recognition, and entrance gate control. Various methods have used for making a faithful OCR system, one of which is neural network that is very popular. Among neural networks, multi-layer perceptron is very famous and is used in most applications. Furthermore, the hidden parallelism in it makes it suitable for implementation on many-cores GPUs; hence we use it as our recognition system.

IV. MLP

Neural network is a highly interconnected structure that consists of many simple processing elements or neurons capable of performing massively parallel computations for data processing and knowledge representation [7]. MLP is a kind of neural network that has two or more layers, and there are synaptic weights in feed forward path of the network, and they are updated with back propagation method. One needs to define transfer function, network architecture, and learning law according to the type of problem to be solved [8].

V. JACKET & CUDA

CUDA is software that is developed by Nvidia group and changed the views to GPUs from solely being graphics rendering devices to be a general computing device as well. The CUDA pack that the NVIDIA group provides includes a driver, a toolkit, and a SDK. The GPU device driver is like a runtime system that works as an interconnection between the software and the device. The toolkit provides the libraries that the codes written in CUDA may link to, so that the CUDA extensions onto the C/C++ languages in our program become interpreted by the compiler and also to provide GPU enabled FFT and BLAS instruction use. FFT and BLAS libraries are two important library of the CUDA pack that facilitates use of FFT instructions and basic linear algebra subprograms and contain efficient functions that are written by experts of Nvidia group. The SDK code samples include a lot of examples that ease the process of learning the CUDA programming. For taking advantage from full capability of GPU, the data communication between CPU and GPU must diminish, because the bus bandwidth between CPU and GPU is bounded. Furthermore, the job that is given to the GPU must have high parallelism in itself so that can obtain the full horsepower of GPU architecture. Although the GPU is capable of running many threads in parallel, this execution is done with the help of kernels defined in CUDA program that work on array of large data elements. There are some limitation for these kernels, two of which- that is, allocating memory, and memory transfer, which come from the barrier on the length of kernels and the sum of local memory they use, are of great importance. Jacket minimizes these tasks transparently and yields high GPU/CPU performance for MATLAB applications with minimal effort from the user.

VI. IMPLEMENTATION

A. *Back Propagation Rule*

Various algorithm can be used for training MLP but the back propagation is the most efficient one [9]. Also, the reality that back propagation algorithm is capable of estimating nonlinear relations between inputs and outputs, make them so famous [10]. However, one of the most important drawbacks of back propagation algorithm is its slow and time consuming train procedure. Ergo, in this research we solve this defect by proposing a parallel back propagation algorithm which is optimized based on the architecture of GPUs. The feed forward back-propagation neural network (BPNN) always consists of at least three layers; input layer, hidden layer and output layer [11]. The neural network should first be trained by applying a large number of datasets before interpreting new information. At the first stage of training, input data is applied to the input layer of the network. Then, the output that is made from this layer is applied to the next layer as input data, and this process continues in the next layers until the outputs are computed in the neurons of the last layer. In this layer, the result of the output neurons are compared with the actual outputs and the difference is processed in the backward direction as error, and the error computation in each layer continues until it reaches the first layer. Based on these computed errors weights of the network are updated according to back propagation formula. Depending on the defined expected error, the whole explained process is repeated for all the training pairs available in the training dataset. After training is completed properly, neural network learns the samples and can detect similarities when presented a new pattern, and accordingly tell what the output pattern is.

Fig. 1 illustrates flowchart of a typical two-hidden-layer BPNN model. The activation function is softmax function.

$$f(x) = e^{x_i} / \sum_j e^{xj} , \qquad (1)$$

Where the sum in the denominator is over all the neurons in the same layer, and is the same for all neurons of all layers. Fig. 1 presents the process of feed forward and back propagation with solid and dashed lines respectively. During feed forward operation each input neuron Y_i receives an input signal and sends this signal to the next layer neurons $Z_1, \ldots, Z_m$ in the hidden layer. The net input to the neuron Z_j and its output are [12]

$$z(in)_j = \sum_{i=0}^{m} Y_i w_{ij} , \qquad (2)$$

$$z(out)_j = f(z(in)_j) . \qquad (3)$$

Where w_{ij} is new weight in the first hidden layer and $Y_0 = 1$ is the bias term. Each neuron in the hidden layer computes its activation and sends its result to the output neurons. The net input to the output neuron O_i and its output are

$$o(in)_j = \sum_{j=0}^{k} K_i w_{ji} \text{ , } \quad (4)$$

$$O(out)_i = f(O(in)_i) \text{ . } \quad (5)$$

Where w_{ji} is new weight in the output layer and $K_0 = 1$ is the bias term. In the next step the back propagation procedure should be executed where the errors are computed and are used for updating weights. The output of the network "o" is computed in the feed forward process that is explained above, and is compared with the target value "t" that is expected to be generated in output layer, and the error is computed. The error function that must be minimized is

$$E = .5 \times (t - o)^2 \text{ . } \quad (6)$$

For output units with softmax transfer function the error is

$$\delta_{pj} = \lambda \times o_{pj}(1 - o_{pj})(t_{pj} - o_{pj}) \quad (7)$$

Where t_{pj} is target output for pattern p on node j and o_{pj} is actual output for pattern p on node j. For hidden layer units with softmax activation function the error is

$$\delta^l_{pj} = \lambda \times o^l_{pj}(1 - o^l_{pj}) \sum_k \delta^{l+1}_{pk} \times w^{l+1}_{jk} \text{ , } \quad (8)$$

Where δ_{pk} and w_{jk} are the error and the weights of next layer respectively, and the sum is over the k nodes in the following layer. Finally, the equation for updating weights is

$$w^l_{ij}(t+1) = w^l_{ij}(t) + \eta \delta^l_{pj} o^{l-1}_{pi} \text{ . } \quad (9)$$

Where $w^l_{ij}(t)$ represents the weights from node i to node j at time t in the layer l, η is a gain constant, and δ^l_{pj} is an error term for pattern p on node j in the layer l, and o^{l-1}_{pi} is the output of the neuron in previous layer for pattern p.

Due to the robustness of back propagation algorithm against noise, we can simply use single precision floating point instead of double precision floating point. By so doing, we can take advantage of the 1 TFLOPS of single precision performance of GPUs instead of 300 GFLOPS of double precision one, and we can reach to greater speeds of execution. Furthermore, we are sure that by correctly and meticulously adjusting the back propagation parameters, the network will converge to the global minimum of the energy surface.

Both feed forward and back propagation contain matrix-matrix multiplication $W \times X$ where W and X can be written as below:

$$W = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & & \ddots & \vdots \\ w_{m0} & w_{m1} & \cdots & w_{mn} \end{bmatrix} \text{, } \quad (10)$$

$$X = \begin{bmatrix} x_{10} & x_{20} & \cdots & x_{1p} \\ \vdots & & \ddots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{np} \end{bmatrix} \text{. } \quad (11)$$

Where W is the weight matrix and X is input data in feed forward (Y) and error in back propagation (δ), and n,m,p are the number of neurons in the first layer, the number of neurons in the hidden layer, the number of patterns in training dataset, respectively.

One kind of variations of back propagation technique is batch update that has an effect on convergence speed and is to only update the weights after many pairs of inputs and their desired outputs are presented to the network, rather than after every presentation [9]. We use the batch update rather than pattern update in back propagation so that we can utilize the maximum capacity of GPU and do updates in parallel. We apply the changes after the complete set of training pairs are presented to the network, and the result of whole matrix-matrix multiplication is computed in only some cycles of GPU. This multiplication takes many cycles of CPU, because it has only up to 4 cores and this computation should be done serially while GPU has at least 10 times more core than CPU and can do all computation in parallel.

B. *Designed Neural Network*

It has been proven that any function can be approximated with the help of solely a three layer feed forward multi-layer perceptron which is trained with the back propagation algorithm [13]. It is obvious that the greater the number of layers become, the more parameters the network should learn, and the more complicated the learning process becomes. Furthermore, the more the number of parameters are, the more samples are needed for training the network, and consequently the network is not capable of generalization and tries to memorize the samples which is not acceptable for us. On the other hand, if we design the network with few layers and few neurons in each layer, the network's ability for generalization lessens and cannot approximate complex functions. Besides, a network with few layers or few neurons in each layer put a great burden on the existing neurons, and learning procedure becomes complex and time consuming. Moreover, owing to the great number of pixels in the pictures that makes the input data, the number of neurons in the first layer is correspondingly great (e.g. 3600). If inadequate number of hidden layer neurons is chosen, the network will not be able to learn data samples and consequently we take wrong answers in test stage. Hence, we choose a three layer BPPN with at least 200 neurons in hidden layer to implement our algorithm.

We use the network architecture depicted in Fig. 1 which is consisted of as many neurons equal to the number of input picture pixels, 200 to 2000 nodes in the hidden layer, and as many neurons equal to the number of classes in the output layer. The inputs to the network are the binary value of pixels made the input picture. Learning is achieved through back propagation with momentum equal to 0.9. The initial learning rate is 0.1. Either a mean square error rate of less than 0.001 or maximum number of 1000 iterations is used as a terminating condition.

C. *Details of Implementation*

D. *I) Initial Weights*

Choosing the initial weights has a great effect on the speed of convergence. Moreover, in some cases, choosing in appropriate initial weights can prevent the algorithm to achieve the global minimum. Choosing a very big initial guesses saturates the activation function. This will reduce the derivative, so correction of weights will be very slow. Choosing the weights in [-0.5 +0.5] and [-1 1] domain is suitable. Based on our experimental results, we figure out that the best domain for initial weights is the first domain.

II) Data Forms

The convergence and at least convergence rate is related to data forms. Choosing inappropriate data form not only diminishes the convergence speed, but also can even prevent the problem from being converged. Input data can be either continuous or discrete. However, the convergence rate for discrete data is faster. Likewise, bipolar data and activation functions are preferred rather than binary data and activation functions. Because obviously in binary form, an inactive unit will not play any rule in learning process. By considering all of the aforementioned facts and based on our experiments, we decided to choose the bipolar form with discretized data.

III) Number of Hidden Layers

While increasing the number of internal hidden layers may diminish the number of local minimums and the chance of wrong classification, this augment the computation time. Here there is a tradeoff between the accuracy and precision which must be considered carefully. There is no generalized and well-defined solution for this problem. In some cases the number of hidden layers is equal to the number of train samples and in some other cases this number will be calculated by an exponential relation to the number of inputs. In this research we obtained this number by using trial and error method by testing different combination and choosing the most suitable one.

IV) Momentum

In updating, large factor of weights will result in network to swing and small factor of weights will slow the convergence rate. To solve this problem we used momentum in updating the weights. By doing so, the steps will be bigger. As a result jumping over weights will be more possible which increase the convergence speed.

It is worthy to mention that in this work we are concentrating on parallelizing and speeding up neural network based OCR. In another word, since this work is related to the speeding up the training and testing phases of multi layer perceptron and not to the recognition accuracy, the network design based on the details provided in the paper does not guarantee to overcome the over fitting problem.

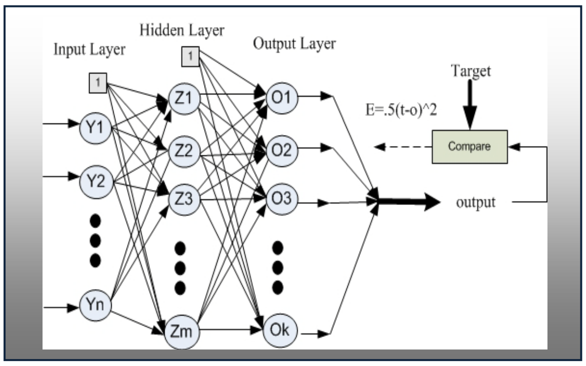


Figure 1. Neural Network structure

E. Computational Hinders

We faced three main problems in the process of back propagation computing which are: NaN (Not a Number) messages from the compiler, subnormal, and saturation. These obstacles are not dedicated to GPU, and sometimes we encounter such problems in CPU computing, but we almost see it in GPU rather than CPU. These unwanted mistakes appear when we make a big neural network with excessive layers which contain many neurons in the input, hidden and, output layers. In normal applications which a big network is not needed to solve the problem, these messages is not seen. As we add the cumulative size in back propagation algorithm, a huge summation must be undertake, and hence floating point overflows, and for this reason we may see NaN for the weights value when debugging our code. As soon as such a NaN produces, it cooperates in the next multiplications and its value will distribute among other weights, making the test result wrong. Lessening the learning rate helps to solve the problem to some extent, but it slows the learning process. To overcome this problem, we changed the softmax function and made it compatible with GPU by dividing both its nominator and denominator into exp(x_{max}); x_{max} is the maximum of inputs to each layer. By doing so, the new equation of softmax function will be obtained which is shown in (12).

$$f(x_i) = \frac{exp(x_j - x_{max})}{\sum_i exp(x_i - x_{max})} \qquad (12)$$

Obviously in (12) the inputs are smaller ergo it is less possible to face an overflow. Second problem caused with miniature values, lead the CPU to label them as subnormals, and the process on these subnormals is very slow. Defining lower cutoff threshold may heal this problem. Next disturbance is the saturation of neurons in the back propagation process that stops any further leaning of this neuron and the weights connected to it are not updated. This phenomenon take place when the value of the neuron output o_{pj} nears to one, and the production $o^l_{pj}(1-o^l_{pj})$ in (7) will produce zero, leading δ^l_{pj} become zero and consequently no more update will occur in (8).

VII. EXPERIMENTAL RESULTS

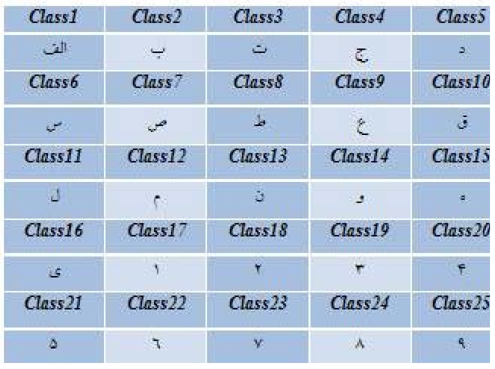
Since there is not a standard database for Farsi characters that are used in Persian license plates, we made a simple handmade database. Our database has 25 classes which is consisted of 9 classes of digits and 16 classes of characters where each class contains 5 samples with different view of camera or illumination. Table II depicts instances of these classes.

We test this algorithm for recognition of number plate of cars in Iran. There are 8 different characters on the number plate of cars in Iran. When our system receive an image of a number plate, the recognition of all units will be start simultaneously. Moreover, not only the recognition of all 8 characters for each number plate is parallel but also, the proposed system is able to investigate several number plates at the same time. Ergo, it makes monitoring of large number of cars at the same time possible.

We copied the achieved weights of neural network in 8 different place of memory. Now, 8 neural networks will work simultaneously and each network is responsible for detecting one of the 8 characters. Note that we trained two different networks. One network is for recognition of letters and another network for recognition of characters. This will separate the process of number recognition and letter recognition, so both number and letter recognition process will be easier, faster and more accurate.

We tested the whole network on four platforms, two of which are laptops and the others are PCs. The result of running our network on the mentioned platforms shows speedup of training and testing procedure of multiple back propagation algorithm only when the number of neurons is greater than a threshold. The configuration of the platforms, the speedup range, and the accuracy of the network in detecting new inputs is presented in Table III. The speedup and the accuracy changes according to the number of neurons in the hidden layer and the value of back propagation parameters. The accuracy is the maximum accuracy that we reached when changing the network parameters.

TABLE II. CHARACTERS IN DIFFERENT CLASS OF DATABSE



Class1	Class2	Class3	Class4	Class5
الف	ب	ت	ج	د
Class6	Class7	Class8	Class9	Class10
س	ص	ط	ع	ق
Class11	Class12	Class13	Class14	Class15
ل	م	ن	و	ه
Class16	Class17	Class18	Class19	Class20
ی	۱	۲	۳	۴
Class21	Class22	Class23	Class24	Class25
۵	٦	۷	۸	۹

VIII. CONCLUSION

Training and testing of neural networks is a time consuming process especially when the volume of data to be processed is great, and that is the case in the system of police car license plate recognition. We implemented a fast system of optical character recognition of Farsi license plate characters using graphics card that are available on most PCs. We took advantage of Accelereyes Jacket software for implementing our system on GPU. With the use of this Jacket the matlab programmer, with knowing some extensions that Jacket adds to matlab, simply writes his program in a user friendly environment of matlab mfile and run it on GPU. So, we easily used the graphics unit interface of matlab for construction of our model and simply run our tests many times on both CPU and GPU. We test our system on 4 computers with different configurations and get the speedup of between 2 and astoundingly 12 factors.

ACKNOWLEDGMENT

We thanks fast processing laboratory of Amirkabir University and CyberSpace Research Institute (former ITRC) that provide us platforms to test and run our system.

REFERENCES

[1] Daniel L. Ly, Volodymyr Paprotski, Danny Yen, “Neural Networks on GPUs: Restricted Boltzmann Machines”, see http://www.eecg.toronto.edu/~moshovos/CUDA08/doku.php?id=project_presentations_reports_source_code;

[2] H. H. Jang, A. J. Park, and K. C. Jung, “Neural Network Implementation Using CUDA and OpenMP,” Computing: Techniques and Applications, pp. 155–161, 2008.

[3] Raghavendra D Prabhu ,“SOMGPU: An Unsupervised Pattern Classifier on Graphical Processing Unit”, Evolutionary Computation, 2008. CEC 2008, 1-6 June 2008,pp. 1011 – 1018, doi: 10.1109/CEC.2008.4630920 .

[4] Daniel Strigl, Klaus Kofler and Stefan Podlipnig, “Performance and Scalability of GPU-based Convolutional Neural Networks”, Parallel, Distributed and Network-Based Processing (PDP) , 17-19 Feb. 2010, pp. 317 – 324, doi: 10.1109/PDP.2010.43 .

[5] F.Bernhard, and R.Keriven, "Spiking neurons on GPUs", International Conference on Computational Science: Workshop on GPGPU, May 2006

[6] J.Moorkanikara Nageswaran, Yingxue Wang, Nikil Dutt, Tobi Delbrueck, "Computing Spike-based Convolution on GPUs", Circuits and Systems, 2009. ISCAS 2009, 24-27 May 2009, pp. 1917 – 1920, doi: 10.1109/ISCAS.2009.5118157 .

[7] Kosko B., Neural networks and fuzzy systems: a dynamical systems approach to machine intelligence. New Delhi: Prentice-Hall; 1994. pp. 12–7.

[8] Simpson PK., Artificial neural system—foundation, paradigm, application and implementations., New York: Pergamon; 1990.

[9] Mutasem khalil Sari Alsmadi, Khairuddin Bin Omar and Shahrul Azman Noah, “Back Propagation Algorithm : The Best Algorithm Among the Multi-layer Perceptron Algorithm”, IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.4 pp. 378-383, April 2009,

[10] Alsmadi, M.K., Bin Omar, K., Noah, S.A., Almarashdah, I., “Performance Comparison of Multi-layer Perceptron (Back Propagation, Delta Rule and Perceptron) algorithms in Neural Networks.”, Advance Computing Conference, 2009. IACC 2009. IEEE International , 6-7 March 2009 , pp. 296 - 299, doi: 10.1109/IADCC.2009.4809024

[11] Tawadrous AS, Katsabanis PD, “Prediction of surface crown pillar stability using artificial neural networks”. International Journal for Numerical and Analytical Methods in Geomechanics, Volume 31, Issue 7, June 2007, pp. 917–931, doi: 10.1002/nag.566

[12] M. Monjezi, H. Dehghani, “Evaluation of effect of blasting pattern parameters on back break using neural networks” International Journal of Rock Mechanics & Mining Sciences, vol. 45, Issue 8, Dec. 2008, pp. 1446-1453.

[13] Haykin Simon, Neural Networks: A Comprehensive Foundation, 2rd ed., Prentice Hall. ISBN 0132733501, 1998,pp. 178 – 278

[14] Suresh, S.; Omkar, S.N.; Mani, V.; , "Parallel implementation of back-propagation algorithm in networks of workstations," Parallel and Distributed Systems, IEEE Transactions on , vol.16, no.1, pp. 24- 34, Jan. 2005.

[15] Casey, R. G.; Jih, C. R.; , "A Processor-Based OCR System," IBM Journal of Research and Development , vol.27, no.4, pp.386-399, July 1983.

[16] Roy, P.P.; Pal, U.; Llados, J.; Kimura, F.; , "Multi-Oriented English Text Line Extraction Using Background and Foreground Information," Document Analysis Systems, 2008. DAS '08. The Eighth IAPR International Workshop on , vol., no., pp.315-322, 16-19 Sept. 2008.

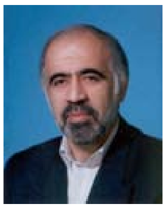
[17] Pal, U.; Roy, P.P.; , "Multioriented and curved text lines extraction from Indian documents," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on , vol.34, no.4, pp.1676-1684, Aug. 2004.

[18] Beg, A.; Ahmed, F.; Campbell, P.; , "Hybrid OCR Techniques for Cursive Script Languages - A Review and Applications," Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on , vol., no., pp.101-105, 28-30 July 2010.

TABLE III. PLATFORMS CONFIGURATION

platform	Configuration							
	CPU processor	*RAM of CPU*	*Operating system*	*GPU model*	*GPU dedicated memory*	*Number of streaming processors*	*Train speedup range*	*% accuracy*
1	AMD Athlon(tm) 64 X2 Dual Core 2.61 GHZ	2 GB	Windows XP 32-bit	NVUDIA GeForce 8400 GS	256 MB	16	0.1 to 1.15	25 - 100
2	Intel(R) Core(TM) i7 Q 720 @ 1.6GHz	4GB	Windows 7 64-bit	NVIDIA GeForce GT 320M	1 GB	24	0.5 to 2.5	25 -100
3	Intel(R) Core(TM) i7 Q 720 @ 1.6GHz	6 GB	Windows 7 64-bit	NVIDIA GeForce GT 330M	1 GB	48	0.6 to 4	25 - 100
4	2 Zeon Quad Core 2.4 GHZ	8 GB	Windows 7 64-bit	NVIDIA TESLA C 1060	4 GB	240	0.8 to 12	25 - 100

Ehsan Arianyan is with High Performance Computing (HPC) Lab. of the Electrical Engineering Department of the Amirkabir University of Technology (AUT). He received his B.Sc. degree from Iran University of Science and Technology in 2008 and his M.Sc. degree from the AUT in 2010. He is currently a PhD student in the AUT and his research areas are high performance computing, cluster computing, cloud computing, and decision algorithms.



Seyed Ahmad Motamedi received his B.Sc. degree in electronic engineering from Amirkabir University of Technology (AUT), Tehran, Iran, in 1979. He received his M.Sc. degree in computer hardware in 1981 and his Ph.D. degree in information systems (computer hardware) in 1984, both from University of Pierre & Marie Curie (Paris VI), France. Currently he is a Full Professor at the Electrical Engineering Department of Amirkabir University of Technology (AUT) and has been a faculty member of this institute since 1984. His research interests include Parallel Processing, Image Processing, Microprocessor Systems, Wireless Industrial Networks, Wireless Sensor Networks, Automation and Biomedical Engineering. Prof. Motamedi was the president of Iranian Research Organization for Science and Technology (IROST) from 1986 to 2001. He is the head of High-Speed Processing Research Center, AUT, Tehran, Iran. He has been the author of several books and has published many scientific papers in international conferences and journals.

Iman Aryanian was born in Iran in 1986. He obtained his B.Sc. degree in electrical engineering from Amirkabir University of Technology, Tehran, Iran in 2008, and his M.Sc. degree from Amirkabir University of Technology, Tehran, Iran in 2010. He is currently working toward his Ph.D. program. His research interests are in the areas of high performance computing, cluster computing, and cloud computing.

Mohammad Motamedi received his B.Sc. degree in electrical engineering from the AUT, Tehran, Iran in 2012. Currently he is pursuing his M.Sc. degree at Electrical Engineering Department of AUT. His main research areas are parallel processing and high performance computing. He is currently with HPC lab of the AUT and is working with Sahisystem Corporation as a system designer.