Pattern Recognition with Spiking Neural Networks

Israel Tabarez Paz, Neil Hernández Gress, and Miguel González Mendoza

Universidad Autónoma del Estado de México
Blvd. Universitario s/n, Predio San Javier Atizapán de Zaragoza, México
itabarezp@uaemex.mx
http://www.uaem.mx/cuyuaps/vallemexico.html
Tecnológico de Monterrey, Campus Estado de México,
Carretera Lago de Guadalupe km 3.5 Col. Margarita Maza de Juarez,
Atizapán de Zaragora, México
{ngress,mgonza}@itesm.mx
http://www.itesm.edu

Abstract This manuscript is focused on some applications of method Spikeprop of Spiking Neural Networks (SNN) using an especific hardware for parallel programming in order to measure the eficience. So, we are interested on pattern recognition and clustering, that are the main problems to solve for Artificial Neural Networks (ANN). As a result, we are going to know the considerations,its limitations and advantages, that we have to take into account for applying SNN. The main advantage is that the quantity of applications can be expanded for real applications linear or non linear, with more than one attribute, and big volume of datas. In contrast, other methods spend a lot of memory to process the information, which computational complexity is propotional to the volume and quantity of attributes of datas, also is more difficult to program the algorithm for multiclass database. However, the main limitation of SNN is the convergence, that tends forward a Local minimum Value. This implies a high dependency on the configuration and proposed architecture. On the other hand, we programmed the algorithm of SNN in a GPU model NVIDIA GeForce 9400 M. In this GPU we had to reduce parallelism in order to increase quantity of layers and neurons in the same hardware in spite of contains 60000 threads, they were not enought. On the otrher hand, the divergence is reduced when the database is bigger for database multiclass.

Keywords: GPU, FPGA, Artificial Neural Networks, Spiking Neural Networks, Image Recognition, Clustering.

1 Introduction

This paper show us some applications of method Spikeprop of Spiking Neural Network (SNN). Firstly, we propose characters recognition looking upon only the five vowels, however this problem can be extended for more letters and numbers.

The result let us to see that this metholology is very good for this applications, but also to know that there are some issues in the future. On the other hand, we apply some real database about patients in a hospital for classification according to some proposed characteristics. In this point, Olaf proposed characters recognition through its pronuntiation [4], however he says that this application is a very difficult because of encoding the datas and spent memory. The most of techniques for image recognition apply known typical techniques as contour detections, image segmentation mathematics tools. Also, traditionals Artificial Neural Networks as Perceptron and Backpropagation have beem used to pattern recognition.

There are others important methologies such as Quadratic Programming (QP) and Sequential Minimal Optimization (SMO) [15], from Support Vector Machine (SVM) [5] they both, also Izhikevich model [7] and Hodgkin – Huxley model [18], from Spiking Neural Networks (SNN) they both. First Support Vector Machine (SVM) has been used for regression, database classification or progression, but its quadratic or cubic computational complexity consumes so much of memory. On the other hand, SNN has been used for speech recognition but there are no reference about applications respect to big databases or image recognition. So, the main motivation of this work starts from knowing the applications areas of SNN in order to measure the efficiency of the method. This methodology was developed in 2003, and was called Spike Response Model (SRM) (Bohte [3], Olaf [4]). Respect to Izhikevich model [7], this is the reduced model of Hodgkin–Huxley integrated of two differential equations that explain behavior of mammal neurons.

On the other hand, parallel programming [12] consists on solving large problems by parts. However, there are several different forms of parallel computing such as bit level, instruction level, and task parallelism. This paper is focused on instruction level as in the figure 1 [10], where it is shown the principle of parallel computing. Problems of parallel programming are solved in language C for CUDA.

This paper is distributed as follows: in section 2 refers to relating works, also section 3 contains aspects related to principles of SNN and its parallelization, the architecture of the network and other considerations are included, also contains the found limitations during programming. On the other side, in section 4 treatment of image is presented. Finally, in section 5 the experiments are shown and section 6 we discuss the conclusions.

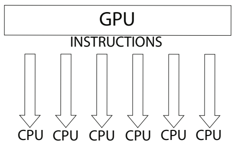

Fig. 1. Parallel programming

2 Relating Works on Spiking Neural Networks

The related works about SNN are shown in the following table 1. The most of authors are focused on the learning time to compare the speed between CPU and GPU, but nobody programmed the optimization QP.

Table 1. State of the Art

AUTHOR	DESCRIPTION OF THE PAPER	DATE
Arash [1]	Compared the speed between CPU and GPU with Izhikevich"s model	2011
Izhikevich [8]	Designed an hybrid model of SNN for numeric methods.	2010
Bhuiyan [2]	Compared the Izhikevich and Hodgkin Huxley model applied to character recognition.	2010
Scanzio [17]	Compared the speed of processing in CUDA of algorithms feedfordward and backpropagation	2010
Xin Jin [9]	Applied a SpiNNaker chip to compare with MatLab. He evaluated the actualization of a neuron, arrangement of entry	2010
Nageswaran [11]	Presented a compilation of the Izhikevich"s models.	2009
Thomas and Luk [17] [19]	Simulated the Izhikevich"s model in a FPGA	2009
Stewart [18]	Applied Runge – Kutta"s method for Izhikevich and Bair [18] and Hodgkin Huxle"s model	2009
Prabhu [16]	Applied GPU for image recognition. He focuses on the degree of parallelism of a problem. The maximum size of image was 256 MB, and in a video memory GPU of 768 MB. As a result, author compares Dual -– Core AMD processor with a Geforce 6150 GPU, and when number of patterns increase, the CPU is linearlly slower than GPU. But when the network size increase then curve is not linear.	2008
Philipp [14]	Implemented SNN in a FPGA to simulate thousands of neurons.	2007
Pavlidis [13]	Applied evolution algorithms in SNN.	2005
Olaf [4]	Theory about SNN and codification of the entries are studies.	2004
Bohte [3]	Compared other algorithms with Spikeprop but author does not show the learning time, but presents differences between traditional ANN and SNN.	2003
Izhikevich [7]	Simulated in MatLab the Hodgkin — Huxley"s model. Also, he executes maximum 10000 neurons and 1 000 000 of sinapsys.	2003

3 Parallel Programming in SNN

The architecture of SNN consists in a set of neurons what contains delays as in the figure 2 (Olaf [4]).

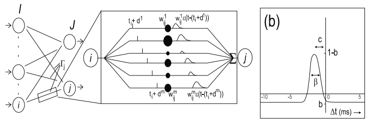

Fig. 2. Conectividad de las SNN

A neuron j, that belongs to a set Γ_j ('pre– synaptic neuron') [4], is fired when the output reaches the threshold θ after receiving a set of pulses in the time $t_i, i\epsilon\Gamma_j$. So, the dynamic of the variable $x_j(t)$ is given by the impulse response $\varepsilon(t)$ where w_{ij} are the weights from conection i to j:

$$x_j(t) = \sum_{i\epsilon\Gamma_j} w_{ij}\varepsilon(t - t_i) \tag{1}$$

The synaptic potential can be defined by the equation (2):

$$\varepsilon(t) = \frac{t}{\tau}e^{1-\frac{t}{\tau}} \tag{2}$$

The constant τ controls the width of the pulse. So, the equation (3) describes the quantity of synaptic conexions of a neuron j:

$$x_j(t) = \sum_{i\epsilon\Gamma_j}\sum_{k=1}^{m} w_{ij}^k y^k(t) \tag{3}$$

A output of a neuron is described by 4:

$$y_j^k(t) = \varepsilon(t - t_i - d_k) \tag{4}$$

Where, d_k is the delay time of a conection k fired from the pre – synaptic neuron to the rise time of the post – synaptic neuron.

Finally, equation 5 represents the output potential $u_j(t)$ for a j nauron:

$$u_j(t) = \sum_{t_j^{(f)}\epsilon F_j} \eta(t - t_j^{(f)}) + \sum_{i\epsilon\Gamma_j}\sum_{t_i^{(g)}} w_{ij}\varepsilon(t - t_i^{(g)} - d_{ij}) \tag{5}$$

Where

$$F_j = \{t^{(f)}; 1 \leq f \leq n\} = \{t|u_j(t) = \vartheta\} \tag{6}$$

And n is the quantity of pulses.

Parallel programming in a GPU involves to solve the problem of configuration of hardware GPU [10]. In case of SNN, is necessary to programme a kernel in three dimensions. In contrast, the simplest configuration in one or two dimensions is used in the perceptron method. In other words, all outputs of neurons per layer can be calculated at the same time because of the simple form of its activation function. However, the activation function of SNN is more complex due to exponential form, which time variable is calculated with mathematic, approximation methods, as a consecuense the outputs of neurons are not always calculated at the same time. In this case, with GPU used we got a near solution because of float presicion, but in case of SVM we need a double presicion to get the best solution.

In a GPU, parameters of the hardware architecture to be considered are threads, blocks and grids [12]. In case of SNN, each block can represent only one neuron, because of computational resources are not enough for database bigger. The main problem of SNN is to calculate the value of threshold in amplitude and time because of this implies that many values of time are needed, as well as, maybe hundreds or thousands columns per block. So, neuron is represented as a grid in tree dimensions, where each row can represent a previous neuron to be added for one output of the following neuron. This method requires good memory resources. Therefore, sometimes it is not recommended to parallel more than hardware features allow you to do.

The arrange of figure 3 represents a configuration of the GPU device in three dimensions. This is a solution for parallelizing SNN algorithm. The cube showed in this figure is only a neuron of a hidden or output layer. There are cubes as neurons are required. Each cube is divided in blocks, what depend on the length of time in the input [4]. All neurons per layer can be calculated in parallel, but a disadvantage is that this procedure requires many resources of memory. So, the time into a block depends on the length between input time and output time.

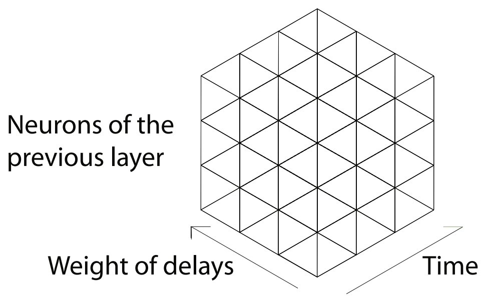

Fig. 3. Parallelization of SNN in 3D

The time is defined in the GPU as a time interval. This model card contains 255 threads por block and 235 blocks, so the total quantity of threads in the card is 60000, aproximately. Configuration of blocks in three dimension is in the equation 7:

$$threadx = \frac{SIZE}{(NN[num\_cap])(NN[num\_cap+1])} \tag{7}$$

Where $threadx$ is the time t; $SIZE$ is threads per block; $NN[num\_cap]$ is the number of neurons in the layer where we want to get the output; $NN[num\_cap+1]$

is the number of neurons in the next layer. We can see that delays are not taking into account for parallel programming, so they are applied sequentially because of memory limitations. Also, if the time exceeds the quantity of threads per block, so we have to add more bloks, but this was encodes sequentially which affects the time processing.

On the other hand, in the figure 4 the vowals in format bmp are shown. But the image can be in other format or it can be a photo. Also, in the figure 5 the architecture of the network is shown, so the quantity of neurons in the input layer is the quantity of pixels of the image. The quantity of neurons of the hidden layer was limited to the hardware.

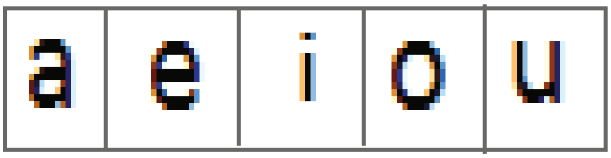

Fig. 4. Image vowals in format bmp (size 5 x 5 pixels)

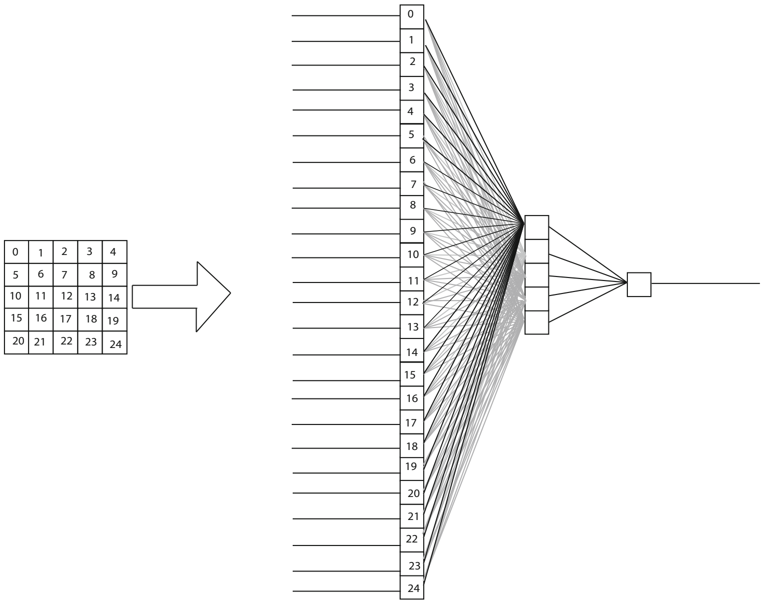

Fig. 5. Architecture of the SNN for image recognition

A big disadvantage of GPU's is the limitation of the hardware, however the parallelism can be reduced or other model of GPU could be considered.

4 Processing of the Image

There are some factors to take into account to get good results, for example the light, color of the objects, noise in the figure 6.

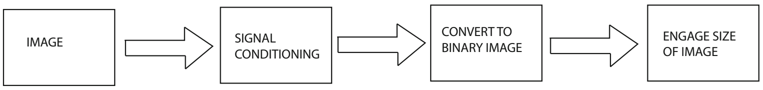

Fig. 6. Processing Image

The size of the image had to be reduced to 5 x 5 pixels because of limitations of the memory of hardware, even thought is not necesary to have so much accuracy because of image proccesing. In this case, the architecture of the network depends on the size of image, so the quantity of pixels represents the quantity of the input neurons. The size of image can not be bigger because of the configuration in the blocks is according to figure 3, so the time axis needs to be large enough which consumes many threads. We used MATLAB to get the right image to be recognized. According to equation (7), it was necessary to reduce the size of image to 5 x 5 pixels, because of the quantity of neurons in the input layer must not to be more than 25 neurons.

5 Experiments

Results of character recognition is presented in this section. In the figure 7 learning stage is presented. It takes approximately two hours to achieve 4 of 5 successes and 9.937205 of quadratic medium error. It would be interesting to achieve more than five neurons in the hidden layer in order to get more efficiency and accuracy. It took aproximately two hours in the learning time. In this case, there are some peaks non desired, so the convergence depends on architecture of the GPU kernel. It would be interesting to increase the quantity of neurons in the layers in order to get better results, but the main problem are the limitations of the GPU and the difficulty to find the optimal architecture of the network and the best values of parameters.

In case of patients of a hospital for classification. We only took a sample 500 patients from 80 000 because it was no possible to take all datas because of memory limitations. This database consist of identify patients with tubeculosis which attributes refers to basical personal information of them.

The attributes are CONSECUTIVE ESTUDY CONTACT, SEX, AGE >20 (18 YES, 20 NO), EXAMINE, EXAMINE 1, EXAMINE 2, EXAMINE 3, CASE (18 YES, 20 NO). The output what we considered was QUIMIOPROFILAXIS (18 YES, 20 NO). In this application, from 500 datas we got 372 success in 21 iterations with 2.3103 73 of quadratic medium error. Time learning was 292237.156250 miliseconds. However, we can select other output attribute according to we need. Also, the name of each patient and ID of tubersculosis was omited of the tranining stage because of simplify.

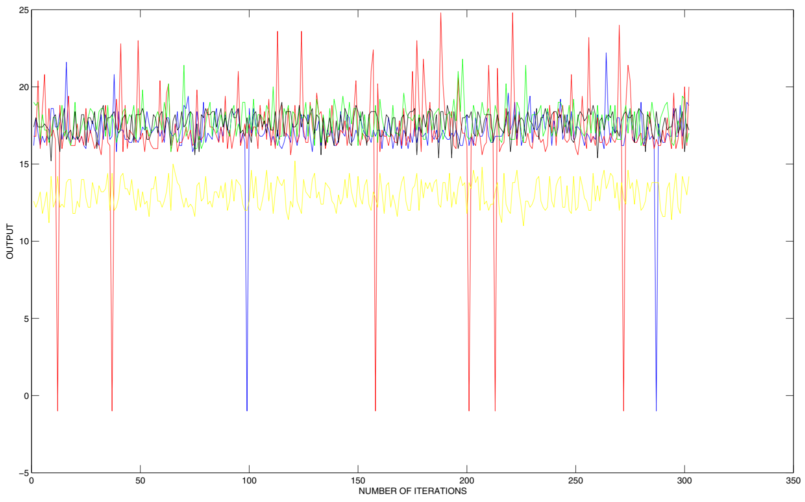

Fig. 7. Learning stage

In the figure 8, we see the convergence of the solution translated to the reduction of the quadratic medium error. However, we can see after fifth iteration the reduction of the error is slower, so after that quantity of successes do not increase as fast as we hope, in fact, solution tends to keep on 372 successes after fifth iteration. This is the main disadvantage of the SNN methology.

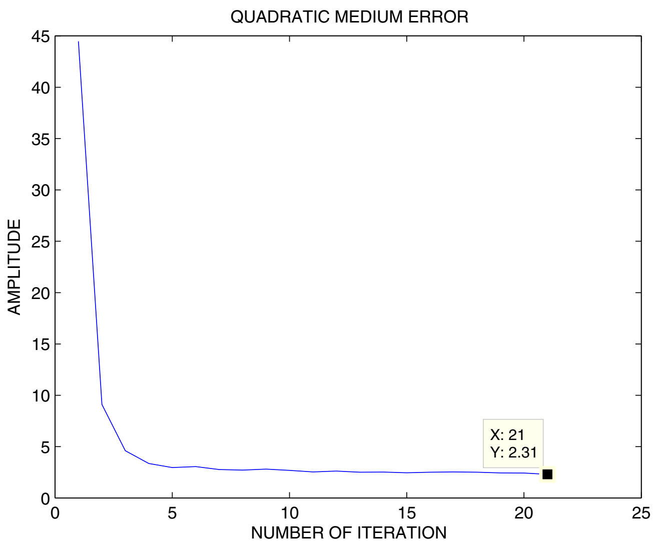

Fig. 8. Learning stage

Finally, we shows efficiency in the table 2.

Table 2. Efficiency of SNN

INPUT DATAS	SUCCESS	EFFICIENCY %
VOWELS	4	80
HOSPITAL DATAS	372	74.4

6 Conclusions

In this paper we conclude that parallelism in SNN increase speed of learning time. However, algorithm SNN has to be improved to converge forward the best solution. We propone to reduce parallelism according to architecture of the card. However, we considered that the best hardware to parallize SNN is FPGA [19], [6] to eliminate problems of copying from host to device and viceversa, what represents to increase learning time. Although, other important problem to get good results is encoding input values in the time. Results and learning time depends on encoding input values and parameters on activation function. Also, with this information is possible to know what applications are the most appropriates for SNN. As a future work, there are other technologies to use for getting good results, as SNN in a FPGA.

References

1. Ahmadi, A., Soleimani, H.: A gpu based simulation of multilayer spiking neural networks. In: 2011 19th Iranian Conference on Electrical Engineering (ICEE), pp. 1–5 (May 2011)
2. Bhuiyan, M.A., Pallipuram, V.K., Smith, M.C.: Acceleration of spiking neural networks in emerging multi-core and gpu architectures. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–8 (April 2010)
3. Bohte, S.M.: Spiking neural networks. Unpublished doctoral dissertation, Centre for Mathematics and Computer Science, Amsterdam (2003)
4. Booij, O.: Temporal pattern classification using spiking neural networks. Unpublished master's thesis, University of Amsterdam (August 2004)
5. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)
6. Fidjeland, A.K., Roesch, E.B., Shanahan, M.P., Luk, W.: Nemo: A platform for neural modelling of spiking neurons using gpus. In: 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2009, pp. 137–144 (July 2009)
7. Izhikevich, E.M.: Simple model of spiking neurons. IEEE Transactions on Neural Networks 14(6), 1569–1572 (2003)
8. Izhikevich, E.M.: Hybrid spiking models. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368(1930), 5061–5070 (2010)

9. Jin, X.: Parallel Simulation of Neural Networks on Spinnaker Universal Neuromorphic Hardware. PhD thesis, University of Manchester (2010)
10. Kirk, D.B., Wen-mei, W.H.: Programming massively parallel processors: a hands-on approach. Morgan Kaufmann (2010)
11. Nageswaran, J.M., Dutt, N., Krichmar, J.L., Nicolau, A., Veidenbaum, A.V.: Efficient simulation of large–scale spiking neural networks using cuda graphics processors. Neural Networks, 791–800 (June 2009)
12. NVIDIA. NVIDIA CUDA C BEST PRACTICES GUIDE DG–05603–001v5.0, dg–05603–001v5.0 ed. (May 2012)
13. Pavlidis, N., Tasoulis, O., Plagianakos, V.P., Nikiforidis, G., Vrahatis, M.: Spiking neural network training using evolutionary algorithms. In: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN 2005, vol. 4, pp. 2190–2194. IEEE (August 2005)
14. Philipp, S., Grübl, A., Meier, K., Schemmel, J.: Interconnecting VLSI spiking neural networks using isochronous connections. In: Sandoval, F., Prieto, A.G., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 471–478. Springer, Heidelberg (2007)
15. Platt, J.C.: Sequiential minimal optimization: A fast algorithm for tarining support vector machines
16. Prabhu, R.D.: Somgpu: An unsupervised pattern classifier on graphical processing unit. In: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 1011–1018, 1–6 (June 2008)
17. Scanzio, S., Cumani, S., Gemello, R., Mana, F., Laface, P.: Parallel implementation of artificial neural network training, pp. 4902–4905
18. Stewart, R.D., Bair, W.: Spiking neural network simulation: numerical integration with the parker–sochacki method. Journal of Computational Neuroscience 27(1), 115–133 (2009)
19. Thomas, D.B., Luk, W.: Fpga accelerated simulation of biologically plausible spiking neural networks. In: 17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM 2009, pp. 45–52 (April 2009)