Anais do IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional (IX CBRN)
Ouro Preto 25-28 de Outubro de 2009 © Sociedade Brasileira de Redes Neurais

PIPELINED ON-LINE BACK-PROPAGATION TRAINING OF AN ARTIFICIAL NEURAL NETWORK ON A PARALLEL MULTIPROCESSOR SYSTEM

Tiago Mendonça da Silva*, Antônio de Pádua Braga*, Wilian Soares Lacerda†

*Computational Intelligence Laboratory - LITC
Department of Electronic Engeneering
Federal University of Minas Gerais
Av. Antônio Carlos, 6627 - Belo Horizonte/MG - CEP 30.161-970 - Brazil

†Department of Computing Science
Federal University of Lavras
PO Box 3037 - Lavras/MG - CEP 37.200-000 - Brazil

Emails: `mendona04@yahoo.com.br, apbraga@cpdee.ufmg.br, lacerda@ufla.br`

Abstract— This work presents an on-chip learning of artificial neural networks in a FPGA multiprocessor system, where each neuron is implemented in a soft-core processor. In order to take maximum advantage of the distributed architecture, a pipelined version of the on-line back-propagation algorithm is used, providing a high degree of parallelism between neuron layers and, hence, a higher speed-up in relation to a sequential implementation.

Keywords— NIOS, FPGA, multiprocessors, backpropagation, pipeline, artificial neural networks.

1 Introduction

Hardware implementation of Artificial Neural Networks (ANNs) learning can be accomplished with high degree of parallelism, since weights within the same neuron layer can be updated independently. In spite of most approaches to learning relying on sequential software implementations, demand resulted from recent higher dimensional problems, such as those related to Bioinformatics and the Internet, has motivated a new wave of interest for high performance physical implementations in the last decade (Zhu and Sutton, 2003; Pethick et al., 2003). The well known back-propagation algorithm (Rumelhart et al., 1986) has being widely used for on-chip learning benchmarking, since it allows exploring the inherent parallelism of the neural network structure. One of the possible approaches to parallel implementation is to pipeline updates and information flow between layers in the forward and backward phases (Gadea and Mocholí, 1999; Gironés et al., 2005).

This work presents an FPGA implementation of a multi-layer ANN on a multiprocessor architecture where each neuron is implemented on a Nios II soft-core processor (Altera, 2007) and shared VHDL components yield communication and data transfer among them. In order to take maximum advantage of the distributed architecture, the Pipelined On-line Back-propagation (PBP) algorithm (Gadea and Mocholí, 1999) is used in such a way that all the neurons of the network work simultaneously, what provides higher speed-up in relation to a sequential implementation.

The algorithm : Without loss of generality, the algorithm will be described for a network with one output and one hidden layer. In order to obtain parallelism among the layers, while the “output-neuron” computes the outputs $h_i[t]$ — provided by the “hidden-neurons” — the hidden-neurons themselves process the next input sample $x_k[t+1]$. In the beginning of each iteration t, the output-neuron feeds back the errors $\delta_i[t-1]$, calculated in the recently completed iteration, and also receives the outputs from the hidden neurons. This exchange of data requires a synchronization mechanism between layers. In Fig. 1, a sequential diagram detailing the tasks of each layer during the learning process is presented.

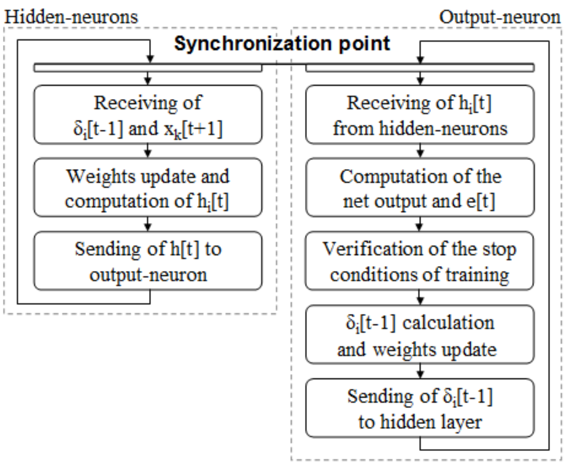
Hidden-neurons
Output-neuron
Synchronization point
Receiving of δi[t-1] and xk[t+1]
Weights update and computation of hi[t]
Sending of h[t] to output-neuron
Receiving of hi[t] from hidden-neurons
Computation of the net output and e[t]
Verification of the stop conditions of training
δi[t-1] calculation and weights update
Sending of δi[t-1] to hidden layer

Figure 1: Sequential diagram of the tasks developed by each neuron during the training process.

According to the algorithm, the weights $w_{ik}[t]$ of the hidden-neurons are therefore updated with a delay of one iteration:

$$\delta_i[t] = e[t] * f'(u_j[t]) * w_{ji}[t] \qquad (1)$$

$$w_{ik}[t+1] = w_{ik}[t] + \eta * f'(u_i[t-1]) * x_k[t-1] * \delta_i[t-1] \qquad (2)$$

where e is the error, $f'(\cdot)$ is the derivative of the activation function $f(\cdot)$, w_{ji} are the output-neuron weights, η is the learning rate, and u_i and u_j are, respectively, the linear outputs of the hidden and output neurons.

2 The multiprocessor architecture

In order to show the performance of the pipelined multiprocessed neural network, the classical XOR problem was chosen. The selected network structure has 2 inputs, 2 hidden neurons and one output neuron. Fig. 2 shows a schematic view of the implemented hardware system. Each neuron is implemented on a Nios II embedded processor (Altera, 2007) and, since they are soft-cores processors, each one is duly configured to attend the needs of the application.

In order to exchange data, the hidden-neurons (CPU 1 and CPU 2) are connected, independently, to the output-neuron (CPU 0) by means of shared VHDL components (forward and backward "memory components"), and semaphore techniques allow the necessary mutual exclusion on their access. An architecture counting on on-chip memory blocks for transfer of data, protected by mutexes, was also implemented, but this approach has resulted on lower performance while demanding more chip space. The shared components are organized in pairs for each hidden-neuron in order to distribute the flux of data among processors, reducing transfer overhead. Only hidden-neurons write data in forward memory components, and, likewise, only the output neuron writes in backward memory components.

Processors and shared components are embedded on a Cyclone II 2C35 Altera's FPGA. However, instructions, data, stack and heap of each CPU are stored in off-chip memories. With the objective of improving parallelism, whereas hidden-neurons share a DDR SDRAM memory, a SSRAM chip — faster — is dedicated to CPU 0, since this last processor has a higher processing overload. Instructions and data on-chip caches (icache and dcache, respectively) are also introduced in each processor in order to minimize access time to the referred external memories, also reducing conflicts among hidden neurons.

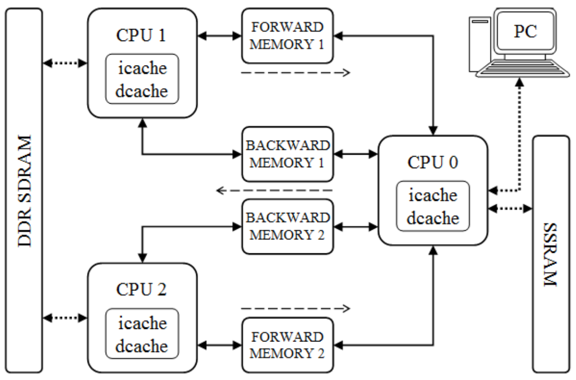

Figure 2: Architecture of a multiprocessor system to process a 2-2-1 neural network:
—— on-chip connections,
· · · · · off-chip connections,
– – – flux of data during training.

System code and configuration settings are downloaded from a host computer to the FPGA and memory chips — located in a Nios II Development Kit (Altera, 2007) — by means of a JTAG UART connection. Soon after training is finished, the same channel is used to save the final weights and network error history — mean square error per epoch — on the host computer, so that the performance of the system can be evaluated.

3 Results

The PBP implementation was compared with a standard sequential back-propagation (SBP) running on CPU 0 — the most powerful among the three CPUs. In order to obtain a comparison between the pipelined and the sequential implementations, two different metrics were used. The first of them is the convergence speed C_s, that is given by the number of epochs completed per time unit, or $C_s = \frac{n_e}{t}$. The second metric is related to the parallelism degree P_d of the implementation and is given by the amount of time a hidden neuron is on hold, within one iteration, waiting for the output neuron at the synchronization point, or:

$$P_d = 100(1 - \frac{t_h}{t})\% \qquad (3)$$

where t_h is the time on hold and t is the total time of the epoch. All the metric results presented were averaged over 10 trials.

The best speed-up — rate between convergence speeds of distributed and sequential implementations — measured was obtained for a system running at 100 MHz (demanding 16210 logic elements and 34560 RAM blocks bytes), wherein CPU 0 differs from the others only by the dcache size (2x larger). The degree of parallelism obtained was $P_d = 0.75 \pm 0.01\%$, what indicates that only in 25% of the time the hidden neurons were on hold. This degree of parallelism resulted on a speed-up:

$$S_{up} = \frac{C_{s(PBP)}}{C_{s(SBP)}} \tag{4}$$

of 2.03 ± 0.03, i.e. the PBP (presenting a convergence speed of 180.16 ± 1.74 epochs/sec) was about 2x faster then the SBP for the current problem. S_{up} is, of course, expected to rise as the number of neurons in the hidden layer increases for solving higher complexity problems. Convergence behaviour can be observed for both PBP and SBP in Fig. 3. Although the delay between updates, given by equation (2), has caused a non-smooth error curve, it has not affected the overall convergence performance.

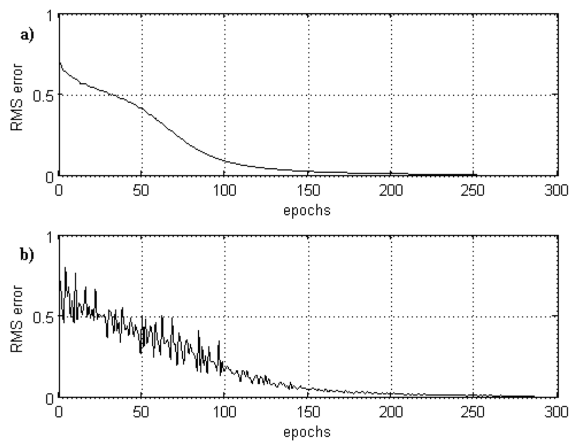

Figure 3: Network error behavior during training:
a - Sequential implementation (SBP),
b - Pipelined multiprocessed implementation (PBP).

4 Conclusions

The implementation presented provides an alternative co-design topology to embed adaptive neural-based systems. A multiprocessor architecture allows better distribution of tasks and an efficient implementation of the PBP algorithm. In addition, when working with soft-cores, system description becomes easier and faster. Providing on-chip learning, the solution is useful for adaptive control and system modeling for real-time applications. Performance and parallelism can be yet improved by distributing the output-neuron tasks among co-processors as more hidden-neurons are eventually necessary. The flexibility of the Nios II architecture and the built-in facilities to deal with concurrent processes paves the way for further more complex solutions as more chip space is available.

Acknowledgment

The authors would like to thank the support from Fundação de Amparo à Pesquisa de Minas Gerais - FAPEMIG.

References

Altera (2007). *Nios II Processor Reference Handbook*, Altera Corporation, 101 Innovation Drive, San Jose, CA 95134, San Jose, USA. Access in November, 2008. Available at: <http://www.altera.com/literature/lit-nio2.jsp>.

Gadea, R. and Mocholí, A. (1999). Forward-backward parallelism in on-line backpropagation, *International Work Conference on Artificial and Natural Neural Networks* pp. 157–165.

Gironés, R. G., Palero, R. C., Boluda, J. C. and Cortés, A. S. (2005). FPGA implementation of a pipelined on-line backpropagation, *Journal VLSI Signal Processing Systems* **40**(2): 189–213. ISSN 0922-5773.

Pethick, M., Liddle, M., Weretein, P. and Huang, Z. (2003). Parallelization of a backpropagation neural network on a cluster computer, *Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems* pp. 574–582.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986). Learning internal representations by error backpropagation, *Parallel Distributed Processing* **1**: 318–362.

Zhu, J. and Sutton, P. (2003). FPGA implementations of neural networks: A survey of a decade of progress, *Lecture Notes in Computer Science* pp. 1062–1066.