# Stream Processing of a Neural Classifier I

**M. Martínez-Zarzuela**
*University of Valladolid, Spain*

**F. J. Díaz Pernas**
*University of Valladolid, Spain*

**D. González Ortega**
*University of Valladolid, Spain*

**J. F. Díez Higuera**
*University of Valladolid, Spain*

**M. Antón Rodríguez**
*University of Valladolid, Spain*

## INTRODUCTION

An *Artificial Neural Network* (ANN) is a computational structure inspired by the study of biological neural processing. Although neurons are considered as very simple computation units, inside the nervous system, an incredible amount of widely inter-connected neurons can process huge amounts of data working in a parallel fashion. There are many different types of ANNs, from relatively simple to very complex, just as there are many theories on how biological neural processing works. However, execution of ANNs is always a heavy computational task. Important kinds of ANNs are those devoted to pattern recognition such as *Multi-Layer Perceptron* (MLP), *Self-Organizing Maps* (SOM) or *Adaptive Resonance Theory* (ART) classifiers (Haykin, 2007).

Traditional implementations of ANNs used by most of scientists have been developed in high level programming languages, so that they could be executed on common *Personal Computers* (PCs). The main drawback of these implementations is that though neural networks are intrinsically parallel systems, simulations are executed on a *Central Processing Unit* (CPU), a processor designed for the execution of sequential programs on a *Single Instruction Single Data* (SISD) basis. As a result, these heavy programs can take hours or even days to process large input data. For applications that require real-time processing, it is possible to develop small ad-hoc neural networks on specific hardware like *Field Programmable Gate Arrays* (FPGAs). However, FPGA-based realization of ANNs is somewhat expensive and involves extra design overheads (Zhu & Sutton, 2003).

Using dedicated hardware to do machine learning was typically expensive; results could not be shared with other researchers and hardware became obsolete within a few years. This situation has changed recently with the popularization of *Graphics Processing Units* (GPUs) as low-cost and high-level programmable hardware platforms. GPUs are being increasingly used for speeding up computations in many research fields following a *Stream Processing Model* (Owens, Luebke, Govindaraju, Harris, Krüger, Lefohn & Purcell, 2007).

This article presents a GPU-based parallel implementation of a Fuzzy ART ANN, which can be used both for training and testing processes. Fuzzy ART is an unsupervised neural classifier capable of incremental learning, widely used in a universe of applications as medical sciences, economics and finance, engineering and computer science. CPU-based implementations of Fuzzy ART lack efficiency and cannot be used for testing purposes in real-time applications. The GPU implementation of Fuzzy ART presented in this article speeds up computations more than 30 times with respect to a CPU-based C/C++ development when executed on an NVIDIA 7800 GT GPU.

## BACKGROUND

Biological neural networks are able to learn and adapt its structure based on the external or internal information that flows through the network. Most types of ANNs present the problem of *catastrophic forgetting*. Once the network has been trained, if we want it to learn from new inputs, it is necessary to repeat the whole training process from the beginning. Otherwise, the ANN would forget previously acquired knowledge. S. Grossberg developed the *Adaptive Resonance Theory* (ART) to address this problem (Grossberg, 1987). Fuzzy ART is an extension of the original ART 1 system that incorporates computations from *fuzzy set theory* into the ART network, and thus making it possible to learn and recognize both analog and binary input patterns (Carpenter, Grossberg & Rosen, 1991).

GPUs are being considered in many fields of computation and some researchers have made efforts for integrating different kinds of ANNs on the GPU. Most research has been done for implementing *Multi-Layer Perceptron* (MLP) taking advantage of the GPU performance in matrix-matrix products (Rolfes, 2004) (Oh & Jung 2004) (Steinkraus, Simard & Buck 2005). Other researchers have used the GPU for *Self-Organizing Maps* (SOM) with great results (Luo, Liu & Wu, 2005) (Campbell, Berglund & Streit, 2005). Bernhard et al. achieved a speed increase of between 5 and 20 times simulating large networks of *Spiking Neurons* on the GPU (Bernhard & Keriven, 2006). Finally, Martínez-Zarzuela et al. developed a generic *Fuzzy ART* ANN on the GPU achieving a speed up higher than 30 over a CPU (Martínez-Zarzuela, Díaz, Díez & Antón, 2007).

Commodity graphics cards provide a tremendous computational horsepower. NVIDIA's GeForce 7800 GTX GPU is able to sustain 165 GFLOPS against the 25.6 GFLOPS theoretical peak for the SSE units of a dual-core 3.7 GHz Intel Pentium Extreme (Owens, Luebke, Govindaraju, Harris, Krüger, Lefohn & Purcell, 2007). Newest generation of graphics cards, like NVIDIA Geforce 8800 Ultra, or AMD (ATI) Radeon HD 2900 XT, can give a peak performance higher than 500 Gflops and 100 GB/s peak memory bandwidth. Graphics cards manufacturers have recently discovered the field of high performance computing as to be a target market for their products and are providing specific hardware and software to couple with enterprises and researchers heavy computational requirements.

## FUZZY ART NEURAL NETWORK STREAM PROCESSING

This article describes a parallel implementation of a Fuzzy ART ANN using a *stream processing model*. In this uniform parallel processing paradigm a series of computations, defined by one function or *kernel*, are made over an ordered set of data or *stream* on a *Single Instruction Multiple Data* (SIMD) basis. The main restriction of the model is also one of the reasons it can provide large increases in performance and a simplified programming model: operations on each stream element are independent, allowing the execution of the kernel on different hardware processing units simultaneously, and avoiding stalls that could occur because of inter-units data sharing.

GPUs used to have two types of programmable processors, namely *vertex* and *fragment* processors. Both kinds of processors were devised to operate on four component vectors, as the basic primitives of 3D computer graphics are 3D vertices in projected space (*x*, *y*, *z*, *w*) and four component colors (*red*, *green*, *blue*, *alpha*). Both vertex and fragment units could be used to execute a *kernel* over a *stream of data (Stream Processing)* and are programmed using *shaders* that can be written using high level languages as Cg (Randima & Kilgard, 2003), GLSL or HLSL. Latest generation of GPUs, like nVIDIA GeForce 8800 GTX, do not include fragment of vertex processors, but unified *Stream Processors* (SPs): generalized floating-point scalar processors capable of operating on vertices, pixels, or any manner of data. These new GPUs can be programmed using CUDA *(Compute Unified Device Architecture)* Toolkit from nVIDIA. CUDA is a promising new software development solution for programming GPUs, simplifying software development by using the standard C language. Before CUDA was launched programming GPUs for *General Purpose* computation (GPGPU) involved translating algorithms into graphics terms (Harris, 2005). Other companies like *Rapidmind* are developing easy-to-program APIs that use just-in-time (JIT) compilers for translating source code into a format that will work on several system's hardware (GPU, Cell or an x86 CPU). Arrays of data can be uploaded from the CPU to the GPU memory and stored in *textures*. RGBA textures can be used to store 4 floating point data per texture unit *(texel)*. Data is modified along the *graphics pipeline* and then written to the *frame-buffer* memory or rendered to a