Hardware Implementation of the Neural Gas

Fabio Ancona, Stefano Rovetta, and Rodolfo Zunino

DIBE - Department of Biophysical and Electronic Engineering - University of Genoa
Via all'Opera Pia 11a - 16145 Genova - Italy - Email: {ancona, rovetta, zunino} @ dibe.unige.it

Abstract—**The paper presents a hardware implementation of the Neural Gas (NGAS) algorithm. The NGAS is based on vector quantization and is applied to very low bit-rate video compression. The algorithm exhibits interesting properties that can be exploited in an HW realization. The modular structure provides inherent parallelism and can therefore be regarded as an open architecture. The neuroboard interfaces to a PC through a standard ISA bus. The novelty of the proposed solution lies in providing a PC-based configurable HW support for VQ training joining affordable costs with satisfactory effectiveness. Simplicity and easy control for HW tests and SW development represent the basic advantages of the overall approach.**

1. Introduction

VQ algorithms can be implemented with easy structures, but, they involve a high computational cost in repeating the same computation for each vector of the codebook: these are ideal conditions for a parallel implementation.

The paper shows hardware architectures for implementing VQ-based neural networks. In particular, the Neural Gas (NGAS) algorithm [1] is proposed, as it exhibits interesting properties that can be exploited in an HW realization. A PC-based implementation was chosen mainly for its flexibility and relatively low development cost. The target architecture is a board interfacing to the PC through a standard ISA bus. The board operates as a "NGAS coprocessor" performing codebook training independently of the hosting system; the modular structure provides inherent parallelism and can therefore be regarded as an open architecture. The board can also operate (after training) for run-time functioning and specific support for videophone applications. Infact, the overall architecture has been designed for high performance image coding, which can apply effectively to both videophone applications and digital TV signal processing [2].

The novelty of the proposed solution lies in providing a PC-based configurable HW support for VQ training joining affordable costs with satisfactory effectiveness. Simplicity and easy control for HW tests and SW development represent the basic advantages of the overall approach. The structure's performance is limited by the PC bus speed (which drives the on-board clock (8MHz)), by the complexity of the programmable control logic and by using one DSP.

2. The "Neural Gas" Model

Vector quantization is the process of approximating a large data set of multidimensional data (e.g., image blocks for image compression) by a reduced number of "prototype" vectors, obtained by clustering several, similar data into one prototype. This approximation resembles that used in scalar quantization, and proceeds by minimizing some error function (usually, mean square error).

The Neural Gas (NGAS) algorithm, developed by Martinetz et al. [1], is an iterative algorithm to train a set of prototypes. At each iteration, a sample datum is received and prototype vectors are ordered according to their distances from the input sample. Prototypes are then adjusted according to their positions on the ordered list: closer vectors undergo larger modifications. The intensity of the adaptation steps and the width of each vector's neighborhood decrease during training, thus providing a stabilization mechanism, also present in similar algorithms (including Kohonen's SOMs [3]). The NGAS training algorithm can be outlined as follows:

1. Set W = a set of K randomly initialized prototypes (k=1...K); set I = a fixed number of iterations.
2. Repeat for i = 1 to I :
   2.1. Input a sample vector $\mathbf{x}$.
   2.2. Compute the distance $d_k = \| \mathbf{x} - \mathbf{w}_k \|$ from each prototype $\mathbf{w}_k$.
   2.3. Sort the list of prototypes according to d_k.
   2.4. Compute the adaptation step $\Delta\mathbf{w}_k$ for each prototype $\mathbf{w}_k$.
   2.5. Apply adaptations to each prototype.
3. Output the set of prototypes W.

0-7803-4122-8/97 $10.00©1997 IEEE

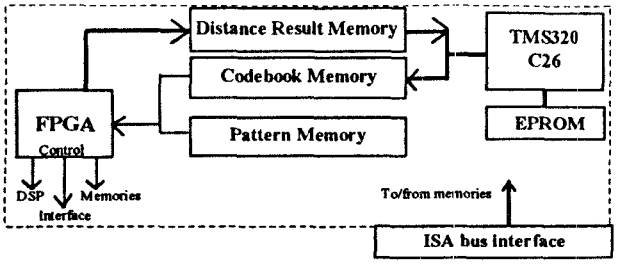

Fig. 1 - Schematic representation of the board components.

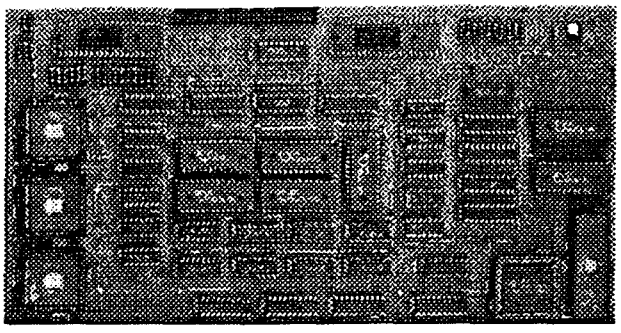

Fig. 2 - Picture of the neuro-board

This procedure exhibits interesting properties that can be exploited in an HW realization. The simple rule provides a uniform coverage of input samples with the available number of prototypes, thus maximizing the representation consistency. Although there is no theoretical proof of convergence, the algorithm has often been shown to have notable advantages over similar models.

3. "NGAS"- Hardware Implementation

A specific feature of the NGAS algorithm [1] guarantees the existence of such an initialization that prototypes always lie in a bounded region, provided that input values are themselves bounded (which is always the case in practice). This is very important when one needs to assess a priori the dynamic range of a stored quantity.

VQ models prove very effective but exhibit the drawback of a huge computational cost due to both the number of processed examples and their dimensionality. This critical issue motivates the research for dedicated architectures to support standard computing platforms.

3.1. Hardware architecture and functioning

The *neuro-board* (Fig.1 and Fig.2) can operates in three different modes:

- training (off-line);
- feedforward phase (run time);
- board testing.

The first functioning mode, after initial memory loading of codevectors and patterns, works out the overall training steps of the algorithm proposed, changing the Euclidean-distance computation into the easier Manhattan-distance computation ($|x - w_k|$ instead of $\|x - w_k\|$). This algorithm modification derives from the need of decreasing computational complexity, verifying that the quality loss in the training ability is acceptable. Figure 3 shows the performances of the NGAS algorithm both when it uses the Euclidean distance and when it adopts the Manahattan distance. These experiments include both the training phase (the training set is composed of the Tiffany and Peppers images) and the test phase (using the Lena image). One can verify that the decreasing of the computational cost does not influence the training ability of the network, thus confirming the validity of this novel approach.

The second mode of the board is similar to the first one, but does not include the adaptation step (step 2.4 of the training algorithm shown in the previous section): it is used when the network is trained.

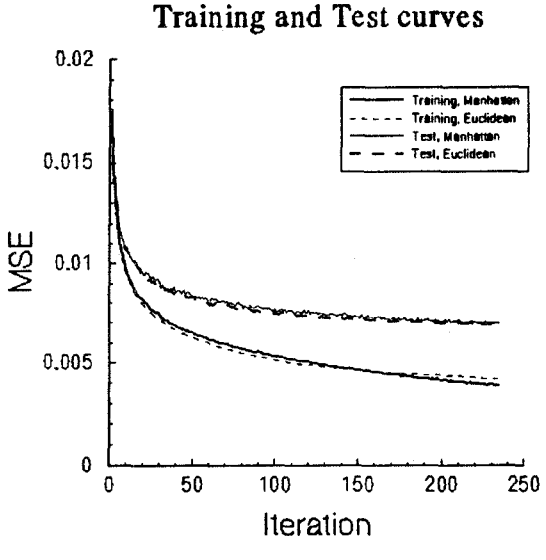

Fig. 3 - Training ability: comparison between the Euclidean distance and the Manhattan distance.

The last mode is used in the hardware-test phase of the board. This system is a visual debugger that allows one to follow the phases of the board functioning. For instance, one can assess if the registers and the memories contain the correct data; therefore, one can verify if the DSP performs all its operations correctly.

A software running on the PC supports the board management. In particular, it supervises patterns and codevectors loading from the PC to the board, PPIs programming, and setting of the board-functioning mode. This management is realized by the PC, loading the *state register* (8 bits) of control signals for supervising each interaction between the PC and the VQ-board. This register plays a crucial role in the board management and allows one to realize a multitasking board by varying the combination of the control signals. In addition, another register, the *control register* (8 bits), is very important for an efficient system functioning: it contains the control signals for the management of each possible interaction among the DSP device and the other components of the overall system (ALU, control units, and PC).

The DSP device is the most important component of the system, as it works out the majority of the NGAS algorithm. For this reason, its communication interface is important to guarantee an efficient functioning of the system. The TMS320C26 is supported by three input ports and two output ports:

Input ports
- The *state register*.
- A register (16 bits) to import the random number for the pattern selection during the training phase.
- A register (16 bits) to receive the operational results of the mathematical coprocessor (Manhattan distances).

Output ports
- The *control register*.
- A register (16 bits) to transfer the final result of the image compression (the index of the winner codevector).

The DSP device has a different computational load, depending on the training mode or the run-time mode that has been set, as summarized in the following:

Training
1. Sorting of distances provided by the ALU.
2. Processing of the adaptation step Δw_k for each prototype w_k (exponential computations), where K is the number of codevectors.
3. Application of adaptations to each codevector ($w_k = w_k + \Delta w_k$).

Run time
1. Sorting of distances (equal to step 1. of the training mode).
2. Transfer of the index of the winner codevector to the PC.

The TMS320C26 is not supported by a floating-point unit (FPU), therefore exponential computations are performed by the DSP's ALU (a Taylor series was chosen to approximate exponential computation). A pre-processing of each floating-point number is necessary to transform it into integer format, and a post-processing, which receives the integer result from the DSP's ALU, must provide the result in floating-point format.

The other computational unit, the ALU, performs the Manhattan distances, using an architecture based on a three-phase pipeline. In particular, the functioning of this structure can be summarized as follows:

Three-phase pipeline:
- Prototype-memory access.
- $\left|x_j - w_j\right|$, $j \in [1, \text{vector size}]$
- $S + \left|x_j - w_j\right|$, where $S = \sum_{i=1}^{j-1} \left|x_i - w_i\right|$, $i \in [1, \text{vector size}]$

It is easy to verify that 68 clocks are necessary for a Manhattan-distance computation between a sample vector, x, and a codevector, w. In addition, the ALU solves the drawback of the different precisions used for the patterns (8 bits per pixel) and for the codevectors (16 bits per pixel), processing each pattern vector in such a way that it becomes a 16-bit vector. There are different techniques to reach this goal and we adopted the following one: the most significant 8 bits are the original pattern, whereas the other 8 bits are equal to the first ones. As a result of this computation, each distance needs at most 22 bits. As the used DSP has a 16-bit local bus, an approximation to distance results is necessary: only the first most significant 16 bits are considered. Each distance is loaded into a 16-bit register (an input port of the DSP), and the processor TMS320C26 can start to work out its process.

The board has two control units, one for I/O operations (I/O-control unit) and the other for the ALU control (ALU-control unit), which are implemented using an FPGA technology. The first unit supports the data-transfer management from the PC to the suitable memory bank on the board, and vice versa. In other words, the input PPI takes data from the bus (now data are available to the board) and then the I/O-control unit manages data transfers into the memory, and vice versa. This supervision is realized by driving opportune control signals toward the PPIs. The other control unit manages both the three-phase pipeline of the ALU and the data-parallel loading from the RAM to the ALU. These tasks are crucial for the overall system performance, and their good

functioning increases the system's efficiency. The parallel access to the memory has been realized by buffering the data bus: this technique consists in isolating (with buffers on the data bus) two portions of the data bus to access both the prototype memory and the pattern memory simultaneously (Fig. 4). This strategy is possible, even though the two memories can be addressed separately.

The overall architecture is structured for parallel processing. As a matter of fact, the two operational units (ALU and DSP) work at the same time: while the ALU performs the current distance, the DSP works out the sorting of the distances already provided. In addition, while the DSP works out steps 2. and 3. of its training phase, another pattern will be read by the ALU controller and the ALU can begin its distance computations. If the board is set in the training mode, the training process will be iterated a number of times set in the initial phase of the system. The main board feature is the adopted configurable hardware solution based on a flexible and cheap PC-hosted implementation. In addition, simplicity and easy control for HW tests and SW development represent the basic advantages of the overall approach.

4. Experimental Results

We tested the VQ-board in the videophone and low bit-rate image coding technical domains. As a matter of fact, the board processes image subblocks with 64 pixels each. In the current prototype version, the number of such samples may vary in the range [256,1024], whereas the codevectors can be at most 256, which is an acceptable number for technical purposes [3][4]. Gray-level pixels are represented with an 8-bit precision, whereas a 16-bit precision allows the codevectors to span the image space more accurately.

Some experimental results on the system's performances mainly concern the critical computational step of the application (computation of a vector distance). Results show that this phase is completed in 8μsec, using a power voltage of 5V, and that the full-operation absorbed current amounts to about 2A, mainly required by the FPGA.

The performances of the overall structure are limited by the mathematical-coprocessor complexity. Simulations based on ALTERA MAX+PLUSII advised a clock frequency of 9MHz. For this reason, the on-board clock is driven by the PC bus speed (8MHz). Under these conditions, the VQ-board completes a feedforward phase (image compression), obtaining a compression of 2 images per second. It is worth pointing out that the DSP device used (TMS320C26) works at a maximum clock frequency of 40MHz; therefore, it stimulates an implementation with advanced technologies, for overcoming the drawback of the low frequency of the on-board clock.

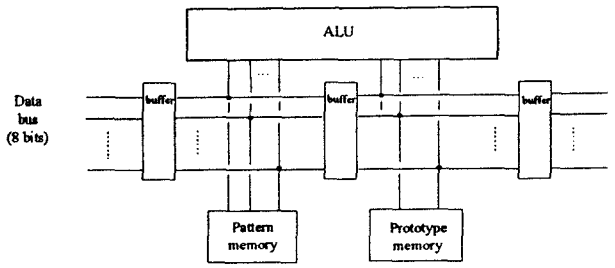

Fig. 4 - Scheme of the parallel data-bus access.

5. Conclusions

We have described an architecture for the implementation of algorithms based on vector quantization. In particular, we have proposed a neural approach (Neural Gas) and presented its hardware realization. The architecture is parallel and composed of two computational units that work out independently. The performances obtained are not excellent, as the technologies used are not advanced. However, the proposed system may be further developed and has been realized at low cost. Future research will involve the use of more powerful devices and the inclusion of twin processors on a single board, yielding a truly parallel computing ability. This will result in a technically significant system providing real-time performances in adaptive videophone applications. To this end, the availability of an integrated development system for high-performance DSPs seems to play a key role.

6. References

[1] Martinetz T., Berkovich S.G., Schulten K.: ' "Neural Gas" network for vector quantization and its application to time-series prediction', *IEEE Trans. on Neur.Net.*, 1993, vol.4, No.4, pp.558-569.

[2] Ancona F., Rovetta S., and Zunino R.: 'Hardware Architectures for Vector Quantization in Very Low Bit-Rate Image Coding', *1996 Int. Workshop on the HDTV*, Oct. 1996.

[3] Anguita D., Passaggio F., Zunino R.: 'SOM-based Interpolation to Image Compression', *Proc World Congr. on Neur.Net. WCNN95*, Washington, vol. I, pp.739-742, July 1995.

[4] Ridella S., Rovetta S., Zunino R.: 'Generalization-based approach to plastic vector quantization' *Proc.World Congr.on Neur.Net. WCNN'95*, Washington, vol. I, pp.505-508, July 1995.