Parallel Architectures for Vector Quantization

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 describes a parallel implementation of neural networks based on vector quantization. A toroidal-mesh topology has been used to assess the overall approach. A theoretical analysis of the modular system's efficiency is presented. The final application goal is a lossy compression of high-dimensional data for low bit-rate communications. Experimental results on a significant testbed shows a remarkable encrease of the system's performances. In addition, the fit between predicted and measured efficiency values confirms the validity of the overall theoretical model.**

1. Introduction

Real-time signal and image processing applications require a very high computational power. For this reason, parallel architectures are often considered to fit the characteristics of these algorithms [1,2]. This design approach is even more useful for the image processing based on neural techniques, as they involve a massive computational load in their neural training process. Neural networks based on vector quantization (VQ) are often considered in image compression domain since they allow a high compression ratio and a good image quality [3-4]. The VQ algorithms can be implemented with easy structures, but, however, they require a high computational cost involved in repeating the same computation for each vector of the codebook. This is the ideal condition for an implementation using special-purpose VLSI processors with a high degree of modularity and local interconnections for data transfer [5].

The paper describes a methodology to implement a neural algorithm for vector quantization, the Neural Gas [6], on parallel HW. In particular, the proposed design methodology for VLSI signal processing architectures has been developed and evaluated by using a toroidal mesh of transputers as a convenient case study of concurrent host architectures. The final application goal is a lossy compression of high-dimensional data for low bit-rate communications. The high computational load of the neural training process and the technical importance of the specific application motivate the search for a highly efficient parallel implementation of the quantization method.

2. The Parallel-NGAS Implementation

The Neural Gas algorithm is an iterative algorithm to train a set of prototypes, and its features fit an SIMD implementation. At each iteration, a training pattern is presented and prototype vectors are ordered according to their Euclidean distances from the input sample. Prototypes are then adjusted according to their positions on the ordered list: closer vectors undergo larger modifications. The intensities of the adaptation steps and the width of each vector's neighbourhood decrease during training, thus providing a stabilization mechanism, also present in similar algorithms (including Kohonen's SOMs [7]).

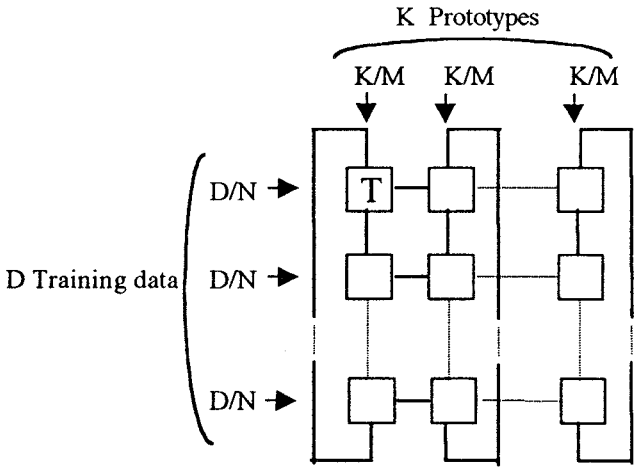
2.1. Architecture

Transputers can be considered as VLSI building blocks to implement massively concurrent architectures. For these reasons, we have developed and evaluated the proposed parallel approach to VLSI signal processing architectures, using a toroidal mesh of transputers (Fig. 1) as a convenient case study of concurrent host architectures. In addition, the choice of using transputers has been encouraged by their high structural flexibility, which allows systems to be designed in compliance with target applications, and by their cost, which is lower than that of commercial workstations. On the other hand, higher development costs demand an accurate architecture design, and the data-allocation strategy and the organization of processors play crucial roles for a system's effectiveness [8].

2.2. Data-allocation Strategy

A straightforward and effective data-allocation method is to split the data set into N subsets and to map them into

0-7803-4122-8/97 $10.00©1997 IEEE



N=No. of mesh rows; M=No. of mesh columns;
D=No. of training samples; K=No. of prototypes.

Fig. 1 - The mesh architecture and the reated data-allocation strategy.

the mesh rows (Fig. 1). As a result, each row is entrusted with the training of one N^{th} of the entire training data set. Conversely, the mutual topological independence of neurons makes it possible to partition the prototype set into as many subsets as the mesh columns. The number of rows and columns are not fixed and can be changed according to the number of available processors.

The above allocation approach has important consequences on the actual algorithm implementation and its efficiency. The system's run-time kernel is arranged in a state machine as follows:

1) Compute the *distances* between available samples and local prototypes by the Euclidean distance.
2) *Sort* prototypes by adopting the following parallel strategy:
   - each mesh row computes the sort computation of the total prototype set;
   - each processor sorts its local portion of neurons (prototypes);
   - a central column of the mesh is chosen to merge the sorted local prototype portions by means of row-wise communications;
3) *Update* prototypes as follows:

$$w_k = w_k + (\Delta w_k^{(local)} + \Delta w_k^{(upper)})$$

where K is the k^{th} prototype of the local neuron portion, $\Delta w_k^{(local)}$ is the local adaptation step, and $\Delta w_k^{(upper)}$ is the adaptation step computed by the processor of the previous row and corresponding to the same column. With this technique, the training contributions of the patterns belonging to the other data-set portions are propagated in the network, and this demonstrates the consistency of the overall parallel approach.

4) *Send* the adaptation step of each neuron to the next row in the mesh.

This approach has several specific features enhancing a parallel performance: the computation-intensive phase, namely, the working out of distances, is performed entirely at the local level, thus yielding the maximum efficiency. Likewise, the vector-adjustment step does not involve any inter-processor communication.

3. Theoretical Analysis

This section presents a theoretical analysis of the efficiency of the overall proposed structure. This analysis makes it possible to derive an analytical expression for the prediction of the system's performance. The following notations and conventions are used:

- P = number of processors;
- τ = time required to transmit a data block, whose size is 4 bytes;
- τ_{sum} = time to perform a floating-point sum;

3.1. Communication overhead

The run-time execution of the proposed application involves two different types of communications:

1) Horizontal transmissions during the *sorting* phase:

$$T_c^{(s)} = \sum_{i=1}^{M-1} (i \cdot 2 \cdot \frac{K}{M} \cdot \frac{D}{N} \cdot \tau_{dist}) =$$

$$\frac{2KD(M-1)}{N}\tau \quad , \quad M \geq 2 \qquad (1)$$

where $\tau_{dist} = 2\tau$ is the time to transmit values of the vector index and the scalar distance (each of them is a data block of 4 bytes). The operator $\sum$ points out the data flow, from the M^{th} column toward the first column: the data flow increases proportionally for each column crossing. The costant term $\frac{K}{M} \cdot \tau_{dist}$ of expression (1) is the data-set portion by which each processor increases the overall data flow. $\frac{K}{M}$ is the neuron-set portion allocated on each processor; the costant 2 of the expression (1) points out the double wave of the data flow (forwards and backwards), as final vector position must be returned to each proces-

sor. The term $\frac{D}{N}$ is the pattern-set portion allocated to each processor: it indicates the number of data flows involved during the overall application (training phase of the neural network proposed).

The real overall transmission time involved in the sorting phase, $T^{(real)}$, is upper bounded by $T_c^{(s)}$; actually, we have simplified a theoretical expression by assuming that the first column is the manager one. This simplification increases the real communication overhead, as the horizontal data flow is single and not more split into two flows, both converging from the most external columns toward the central column at the same time.

2) Vertical transmissions during the *sending* adjustment:

$$T_c^{(\Delta w)} = \frac{D}{N} \cdot \left\{ \frac{K}{M} \cdot m \cdot \tau \cdot [2 + (N \bmod 2)] \right\} \quad (2)$$

where m is the vector size

If N is even, then the vertical transmissions are parallel and they are performed in two steps:

Step 1: data communications between the 1st and 2nd rows, between the 3rd and 4th rows, and so on, until between the $(N-1)^{th}$ and the N^{th} rows, at the same time;

Step 2: data communications between the 2nd and 3rd rows, between the 4th and 5th rows, and so on, until the N^{th} and the 1st rows, simultaneously.

Otherwise, if N is odd, these transmissions require three steps, where the 1st and 2nd steps are analogous to the previous case for the first $N-1$ rows, while the 3rd step involves communications between the N^{th} and the 1st rows.

3.2. Computational timings

At run time, each processor performs three different computations, involving the following three timings:

- *distance* phase:

$$T_{par}^{(d)} = \frac{D}{MN} \cdot \tau_d \quad (3)$$

where D/N is the size of the pattern-set portion: this value is divided by M, as only the local data contribution is considered.

- *sorting* phase:

$$T_{par}^{(s)} = T_{local}^{(s)} + T_{global}^{(s)} = \frac{D}{N}\left(\tau_s^{(l)} + \frac{K}{M}(M-1)\tau_m\right) \quad (4)$$

The time involved in the *sort* phase is composed of two contributions, that is, the time to sort the local codevector portion, $\tau_s^{(l)}$, that is, K/M vectors, and the time to merge the M sorted codevector portions. Let τ_m be the time involved for merging a vector into the global neuron list of the central column.

The first contribution is equal for each processor, whereas the second one has a higher computational load for the processor of the manager column. For this reason, in the effectiveness expression, we consider for each processor the computational cost involved in the manager processors, thus obtaining a default approximation of the system's efficiency.

- *adjustment* phase:

$$T_{par}^{(\Delta w)} = \frac{D}{MN}(\tau_{\Delta w} + mK\tau_{sum}) \quad (5)$$

where $\tau_{\Delta w}$ is the time to compute the vector adjustment step for K prototypes, that is, $w_k = w_k + \Delta w_k^{(local)}$, for $k = 1 \ldots K$. In the above expression, $\tau_{\Delta w}$ is divided by the number of columns, M, though only the contribution of the local prototype-set portion must be considered: this computational cost increases linearly with the number of codevectors. The other expression term, $\frac{mK}{M}\tau_{sum}$, is the time to add the adjustment step obtained in the previous row, $\Delta w_k^{(upper)}$.

2.3. Architecture efficiency

The efficiency of a parallel architecture is defined as the system's speedup over the number of processors used in the network, that is, in other terms, as

$$\eta = \frac{1}{P} \cdot \frac{T_{seq}}{T_{par}}$$

where T_{seq} and T_{par} are the timings for the sequential and the parallel executions, respectively. The timing of the sequential algorithm can be expressed as follows:

$$T_{seq} = T_{seq}^{(d)} + T_{seq}^{(s)} + T_{seq}^{(\Delta w)} = D \cdot \tau_d + D \cdot \tau_s + D \cdot \tau_{\Delta w} \quad (6)$$

where τ_d, τ_s and $\tau_{\Delta w}$ are the timings for performing the distance, the sorting and the adaptation steps of the NGAS algorithm, respectively.

By combining the communication overheads (1)-(2) with the computational timings (3)-(4)-(5), one obtains the timing of the proposed concurrent process:

$$T_{par} = T_c^{(s)} + T_c^{(\Delta w)} + T_{par}^{(d)} + T_{par}^{(s)} + T_{par}^{(\Delta w)} =$$
$$= \frac{2KD(M-1)}{N}\tau + \frac{D}{N} \cdot \left\{ \frac{K}{M} \cdot m \cdot \tau \cdot [2 + (N \bmod 2)] \right\} +$$
$$\frac{D}{MN}\tau_d + \frac{D}{N}\left(\tau_s^{(l)} + \frac{K}{M}(M-1)\tau_m\right) + \frac{D}{MN}(\tau_{\Delta w} + mK\tau_{sum}) \quad (7)$$

One obtains the efficiency expression for the overall system by combining (6) with (7):

$$\eta = \frac{1 + \dfrac{\tau_s}{(\tau_d + \tau_{\Delta w})}}{1 + \dfrac{[2M(M-1) + m(2 + N \bmod 2)]K\tau + M\tau_s^{(I)} + (M-1)\tau_m + mK\tau_{sum}}{(\tau_d + \tau_{\Delta w})}} \tag{8}$$

4. Experimental Results

We evaluated the overall approach using an application testbed consisting in an image-compression task, in which a low bit-rate coding is achieved by VQ encoding. We used a toroidal-mesh architecture composed of 6 transputers (2 columns and 3 rows) of the T800 family, using inter-transputer links operating at 20 Mbit/sec. The compression system processed standard (grey- level) images (8bpp) with 512x512 pixels. All pictures were split into 4096 blocks including 8x8 pixels each. In the experiments, a set of classical pictures were used for the network training, and a different image set for the generalization-based algorithm control.

In the graph in Fig.3, the training and test (the "Lena" picture) costs are plotted versus the number of prototypes used. The curves show that the relative improvement on test data decreases progressively; the fact that the test-cost curve becomes "flat", while the training one keeps decreasing, marks an incipient overfitting and triggers the generalization-based stopping condition. This situation indicates the estimated "best" number of neurons which balances the representation accuracy with the size of the vocabulary. In the case considered, the estimated optimal cardinality of the prototype set lies in the range [190, 230].

The network's performance was obtained on a "validation" picture not used for training nor for cross-validation. Results attained a compression ratio of 42.7, with a PSNR of 28.26 (SNR=22.71, MSE=97.90), indicating the method's notable performance as compared with classical compression techniques (e.g., JPEG).

Figure 4 shows the system's efficiency when it runs in the training mode: it is measured versus the number of neurons and versus the vector size. The training phase has been set for a number of patterns (set of training data) equal to 100 and for a number of global iterations equal to 100 (1 iteration involves the training for the overall pattern set). Experiments involve 16- and 64-vector sizes, as they are considered the most significant in the image-compression domain. Better performances are obtained by a 16-vector size and by increasing the number of codevectors, and this is due to the higher ratio between the compuational cost and the communication one.

Another important verification of the consistency of this research is the correctness of the predicted efficiency (8) as compared with experimental results. For this reason, we measured the time required to transmit a data block (4 bytes), obtaining τ = 7.98 μsec (including both fixed and variable communication costs), the tim-

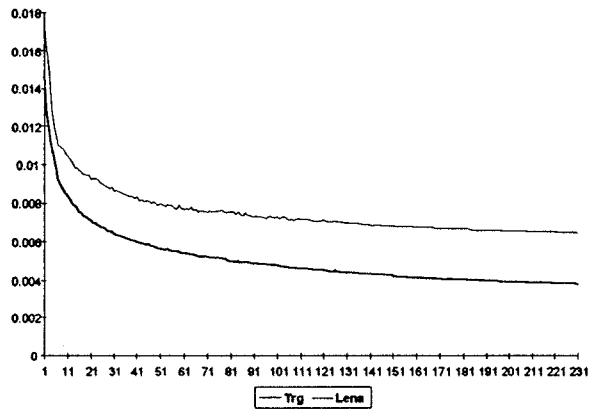


Fig. 3 - Analog-cost curves (x axis = neuron number) of NGAS for image compression

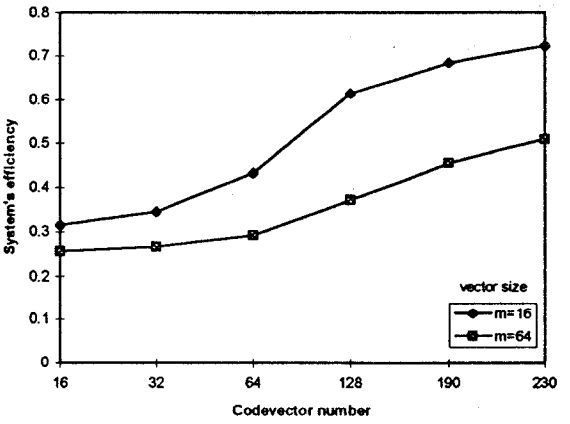


Fig. 4 - System efficiency versus the number of neurons and the vector size.

Table I - Efficiency results

	T_{seq} [sec.]	T_{par} [sec.]	Measured η	Predicted η
K=190	3637.8	1265.9	0.478	0.456
K=210	4255.9	1416.2	0.483	0.500
K=230	4920.2	1570.4	0.509	0.522

ings required for adding a pair of floating-point values amounted to τ_{sum} = 4.29 μsec., and the time for merging a vector distance into the global distance list during the sorting phase was τ_m = 750 μsec. Using these technical values in expression (8), one obtains the efficiency estimates. Table I shows a comparison between predicted and measured values; the fit between experimental and expected values demonstrates the validity of the theoretical model. The comparison involves only the 64-size vectors, as it is the typical vector size in the VQ-based image-compression domain, and the most significant cardinalities of the prototype set (K=190, 210, and 230).

5. Concluding Remarks

Vector Quantization can provide an image-coding schema with a remarkable compression ability, thanks to the codebook-indexing mechanism intrinsic to the quantization process. This advantage is often obtained at the cost of some coarseness and blockness affecting the reconstruction quality. A crucial issue inherent in all these methodologies is the computational cost of training, especially in high-dimensional domains with many training samples.

Therefore, a method for a parallel implementation with high efficiency appears very interesting and useful from a practical perspective, too. In this regard, the paper has presented a general methodology that combines a low-cost machinery with a scalable and effective implementation of the neural model. This represents the basic advantage and the main novel point of the described method.

The current lines of research in this area concern the development of more complex architectures integrating several processors for a real-domain utilization.

6. References

[1] Allen, J.: 'Computer architecture for digital signal processing', *Proc.IEEE*, 1985, pp. 852-873.

[2] Seitz, C.L.: 'Concurrent VLSI architectures', *IEEE Trans.*, 1984, C-33, pp. 1247-1265.

[3] Gray, R.M.: 'Vector Quantization', *IEEE Acoustics, Speech, and Signal Processing Magazine*, Apr. 1984, pp. 4-29

[4] Linde, Y., Buzo, A., and Gray, R.M.: 'An algorithm for vector quantizer design', *IEEE Trans. Commun.*, Jan. 1980, vol. COM-28, pp. 84-95.

[5] Ancona, F., Rovetta, S., and Zunino, R.: 'A parallel Approach to Plastic Neural Gas', *1996 IEEE Int.Conf. on Neural Networks*, June 1996.

[6] Martinez, T. M., Berkovich, S. G., and Schulten, K. J.: '"Neural-Gas" network for vector quantization and its application to time-series prediction', *IEEE Transaction Neural Networks*, 1993, vol. 4, No. 4, pp. 558–569.

[7] Kohonen, T.: 'Self-organization and associative memories', Heidelberger:Springer, 1982.

[8] Pagano F., Parodi G., and Zunino R., 'Parallel implementations of associative memories for image classification', *Parallel Computing*, 1993, vol. 19, No. 6, pp. 667–684.