ELSEVIER

Parallel Computing 23 (1997) 1479–1491

PARALLEL COMPUTING

Transputer-based implementation of distributed associative memories

Fabio Ancona *, Stefano Rovetta, Rodolfo Zunino

Department of Biophysical and Electronic Engineering (DIBE), University of Genoa, Via all'Opera Pia 11a, 16145 Genoa, Italy

Received 29 March 1996; revised 15 November 1996

Abstract

The Distributed Associative Memory (DAM) model operates as a basic device inside an image-classification system, whose accuracy is increased by the restoration ability of the associative mechanism. The paper describes a methodology for implementing the DAM model on transputer-based hierarchical parallel architectures. First, a theoretical analysis reformulates the basic memory model to fit a practical implementation; then, the parallel system's efficiency is defined analytically. Experimental results on a significant testbed confirm the classifier's effectiveness and the consistency of theoretical predictions. © 1997 Elsevier Science B.V.

Keywords: Distributed associative memory; Image-classification system; Transputer-based system; Parallel architecture; Experimental results

1. Introduction

Associative memories are used in several applications of image understanding, as they can enhance the robustness and content-addressability of a recognition system for high-dimensional data processing [1]. Memory models differ in the way they store and recall data. The associative recall process recovers stored information (*datum*) previously associated with an input (*key*); the crucial feature of associative storage lies in the

* Corresponding author.

PII S0167-8191(97)00068-9

ability of the retrieval mechanism to restore incorrect or incomplete information in the addressing key.

This paper considers Kohonen's Distributed Associative Memories (DAMs) [6]. The DAM model is applied to image classification, and its implementation on hierarchical architectures is presented. Several studies on hierarchical structures include the structural features of massively parallel architectures for high-dimensional data processing [2,5,8,13], and the robustness and implementation properties of binary trees [3,14]. The model's ability to perform robust high-dimensional data processing was previously demonstrated in its early development [7]. In later works, DAMs found successful applications in visual-understanding problems, including invariant object recognition [12,15] and autonomous vehicle guidance [10].

The present paper mainly concentrates on evaluating the performance of the parallel implementation. The mathematical framework of the memory model is redefined to satisfy the limited-resource constraints of practical implementations. In particular, the model reformulation is proved to decrease the total memory occupation by orders of magnitude, thus making a practical realization feasible. The higher performance of the parallel approach allows the use of this method for real-time tasks.

As a result, a major novel aspect of the presented research from a theoretical point of view consists in the derivation of a more efficient and practical expression of the basic model. From a practical perspective, the parallel implementation methodology is general and has proved effective in a wide range of working conditions. A calculation of communication overheads is presented and it can be applied to a general application organized according to a general tree topology, in which a data-allocation strategy splits data into 'slices'. For practical applications, transputers provide a suitable machinery to balance expected performances and cost constraints.

The paper outlines the basic memory model and its reformulation in Section 2. Section 3 describes the parallel approach to the memory implementation and the classifier schema, and develops the mathematical model of the parallel structure, from which efficiency performance can be predicted. Experimental results concerning the system's efficiency and classification accuracy are reported in Section 4. Concluding remarks are made in Section 5.

2. The associative model

2.1. Kohonen's basic model

In Kohonen's model, the writing and reading phases are based on matrix multiplications embedding mutual correlations among processed data [6]. The related cognitive framework refers mainly to biological-consistency aspects [7]; however, the overall mathematical formalism can be expressed in terms of simple matrix algebra. The associative system developed in the presented research adopts auto-associativity as a storage schema in which the information used for memory addressing coincides with the information retrieved.

2.1.1. Data structures

The following data structures will be used to implement DAMs for image classification; for simplicity and without loss of generality, square images made up of $N \times N$ pixels will be assumed:

- $\mathbf{M}[N^2, N^2]$ is the memory matrix, resulting from the writing phase;
- $\boldsymbol{p}_i[N^2]$ is a vector holding a prototype (the related image is arranged in $\boldsymbol{p}$ along rows);
- I is the total number of stored prototypes;
- $\mathbf{S}[N^2, I]$ is the prototype matrix, whose columns hold vectors $\boldsymbol{p}_i$, $i = 1, \ldots, I$;
- $\mathbf{S}^t$ is the transposed matrix $\mathbf{S}$;
- $\mathbf{F}[N^2, N^2]$ is the 'forcing stimulus' matrix holding the expected memory output associated with prototypes $\mathbf{S}$.

In autoassociative storage, each forcing stimulus (i.e., the expected recalled information) and the corresponding prototype coincide, hence $\mathbf{F} = \mathbf{S}$.

2.1.2. Memory operation

Memory writing requires the selection of a training set including prototypes $\boldsymbol{p}_i$; training patterns are stored in the memory, $\mathbf{M}$, and coupled with the desired output recall, $\mathbf{F}$. When an image is presented to the system for classification, it is used as a key pattern, $\mathbf{k}$, to address the memory and retrieve information. To this end, the memory matrix is multiplied by the key and a recall vector, $\boldsymbol{r}$, is obtained:

$$\boldsymbol{r} = \mathbf{M}\mathbf{k} \tag{1}$$

In ideal working conditions, the encoding performed by the autoassociative schema is *loss-less*, hence the memory-reading result coincides with one of the prototypes stored in $\mathbf{M}$:

$$\boldsymbol{r} = \boldsymbol{p}_i \quad i \in \{1, \ldots, I\} \tag{2}$$

Thus the memory-training problem can be mapped into a matrix-computation one, in which one has to find the best matrix, $\mathbf{M}$, such that Eq. (2) ideally holds for each i. Kohonen's work on the theoretical model of DAMs shows [7] that, in the case of autoassociative storage, the optimal solution to the training problem can be obtained by a series of matrix multiplications, as follows (prototypes constitute the columns of matrix $\mathbf{S}$):

$$\mathbf{M} = \mathbf{S}(\mathbf{S}^t\mathbf{S})^{-1}\mathbf{S}^t \tag{3}$$

2.2. Model reformulation

The matrix size $\mathbf{M}$ is a critical aspect of Kohonen's memories because, especially for images of 'normal' size, it requires a huge amount of physical memory. In particular, the total memory occupation turns out to be proportional to N^4. As a result, implementing Kohonen's algorithm directly may prove impractical on any computing platform.

In the present paper, this problem is solved by a simple reformulation of the overall memory model, as the resulting model makes the implementation of DAMs feasible on

parallel machinery. Such formalism may appear straightforward; indeed, the reformulation represents a novel point in the context of Kohonen's associative modelling and provides a viable way to the practical use of such devices in high-dimensional data processing. The memory matrix, $\mathbf{M}$, is split into two partial matrixes, $\mathbf{M}_1$ and $\mathbf{M}_2$, which are computed (off line) during training, as follows:

2.2.1. Training

$$\mathbf{M}_1 = \mathbf{S}(\mathbf{S}^t\mathbf{S})^{-1}; \quad \text{size: } [N^2 \times I] \tag{4}$$

$$\mathbf{M}_2 = \mathbf{S}^t; \quad \text{size: } [I \times N^2] \tag{5}$$

2.2.2. Recall

Memory reading is split into two subprocesses: first, the key vector, $\boldsymbol{k}$, is multiplied by $\mathbf{M}_2$ and gives the intermediate vector, $\boldsymbol{v}$; then, the product of $\boldsymbol{v}$ by $\mathbf{M}_1$ yields the final recall, $\boldsymbol{r}$:

$$\boldsymbol{v} = \mathbf{M}_2\boldsymbol{k}; \quad \text{size: } [I \times N^2] \cdot [N^2] = [I] \tag{6}$$

$$\boldsymbol{r} = \mathbf{M}_1\boldsymbol{v}; \quad \text{size: } [N^2 \times I] \cdot [I] = [N^2] \tag{7}$$

Proving that the reformulated memory model is equivalent to Kohonen's original model is immediate:

$$\boldsymbol{r}^{(\text{reform.})} = \mathbf{M}_1(\mathbf{M}_2\boldsymbol{k}) = \mathbf{S}(\mathbf{S}^t\mathbf{S})^{-1}(\mathbf{S}^t\boldsymbol{k}) = \left[\mathbf{S}(\mathbf{S}^t\mathbf{S})^{-1}\mathbf{S}^t\right]\boldsymbol{k} \tag{8}$$

Eq. (8) shows that the recall vector for the reformulated model is equivalent to Eq. (3).

Eqs. (6) and (7) indicate that $\mathbf{M}_1$ and $\mathbf{M}_2$ have an identical number of elements ($I \cdot N^2$), hence the total number of stored elements amounts to $2 \cdot I \cdot N^2$. The advantage over the original memory model (requiring N^4 elements) in terms of physical occupation is notable, especially if one reasonably assumes I to be much smaller than the number of pixels in an image. As compared with Kohonen's formulation, the total memory occupation depends on the number of stored images; this might appear as a drawback in terms of generality. On the other hand, the specific practical improvement holds under standard working conditions; for example, if $I < 1000$ and the involved (small) images include 128×128 pixels, the saving in memory is equal to about one order of magnitude, and may represent a good threshold for the new method's effectiveness (Table 1).

Table 1
Memory occupation

Number of samples	16	160	1600	16000
Memory occupation	2 Mbytes	20 Mbytes	200 Mbytes	2 Gbytes

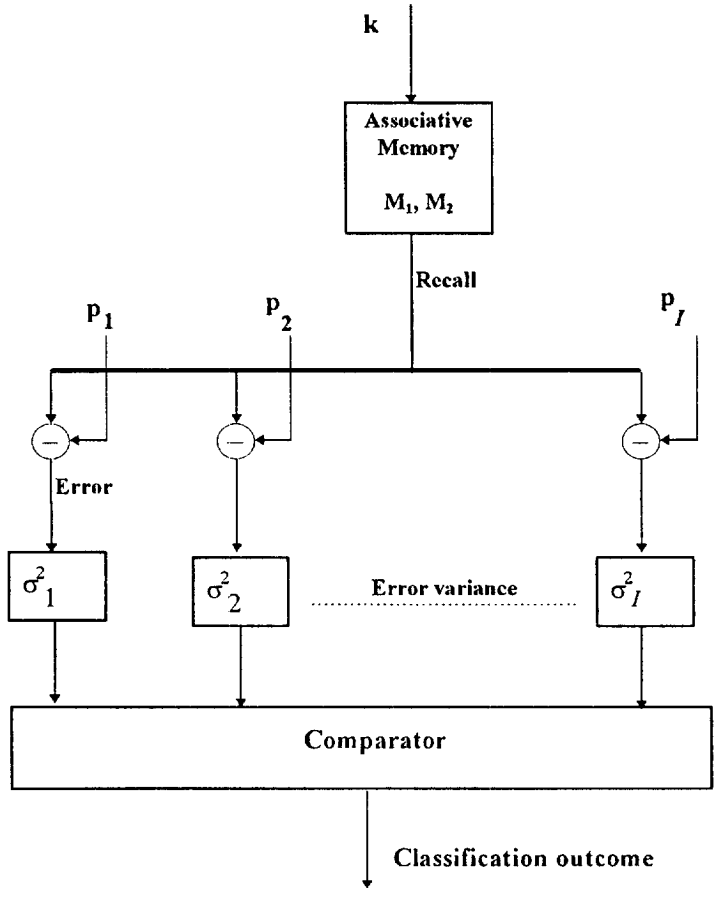

Fig. 1. Schema of the associative classifier.

2.3. Associative classification

In image classification, an unknown input picture must be associated with one of the prototypes stored in the memory. In an associative classifier, the input pattern, **K**, is used as a key for addressing the memory content. Autoassociativity enhances the classifier performance by exploiting the memory's pattern-completion ability, which can restore correct information from noisy or incomplete input keys.

The recognition mechanism is based on a minimum-variance criterion, whose validity was previously verified by using other associative memory models [4,9]. The principle of operation is that the associative device operates as a sort of 'active filter' that recovers the signal-to-noise ratio by using stored information.

The classifier (Fig. 1) proceeds as follows:

- **K** is used to fetch the memory and to derive a recall pattern, r;
- for each prototype, p_i, the mean square error, σ^2, with respect to r is computed;
- the prototype associated with the smallest variance is selected for classification.

The associative classifier is set up *offline* by building the memory device; this is accomplished by storing reference prototypes in an autoassociative way. At run time classification, matching memory recalls with stored prototypes exploits the memory device as a nonlinear, 'active' filter able to restore information. As to the method's robustness, the classification principle has been successfully compared in Ref. [9] with the corresponding optimal schema adopted by classical pattern recognition, which minimizes the MSE.

The specific characterization of the associative classifier's performance goes beyond the scope of this paper; from a general perspective, the apparently surprising result of improving an optimal classifier is explained by the fact that the associative device uses, in the recognition process, some a priori knowledge about stored information *before* issuing the actual classification outcome. These considerations motivate the research on a parallel implementation of the overall system, as it may support real-time performances of the *online* classification task.

3. Parallel implementation

3.1. Parallel architecture and data allocation

The choice of the supporting HW architecture is crucial to eventual applications, and a tradeoff between computational speed and cost constraints should be allowed, especially in practical problems. The implementation of DAMs mainly involves matrix multiplications, whose mathematical formalism somewhat drives the choice toward a parallel approach. To this end, we selected a transputer-based architecture. In this domain, transputers prove inexpensive and feature high flexibility; however, the widely general model reformulation allows one to switch to other and possibly more effective HW supports.

As to the data-allocation strategy and the processor configuration, we adopted a hierarchical network, whose topology is arranged as a tree (Fig. 2). The method used to work out a memory recall is independent of the tree depth and is based on a uniform

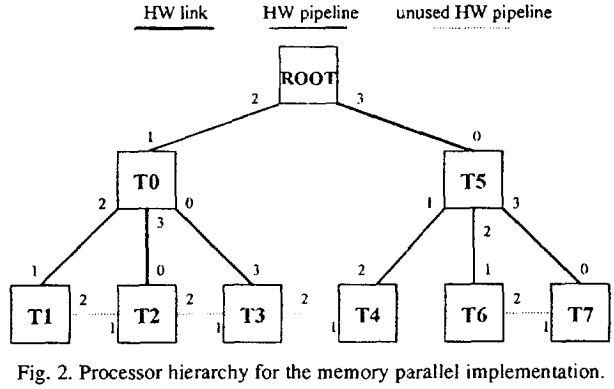

Fig. 2. Processor hierarchy for the memory parallel implementation.

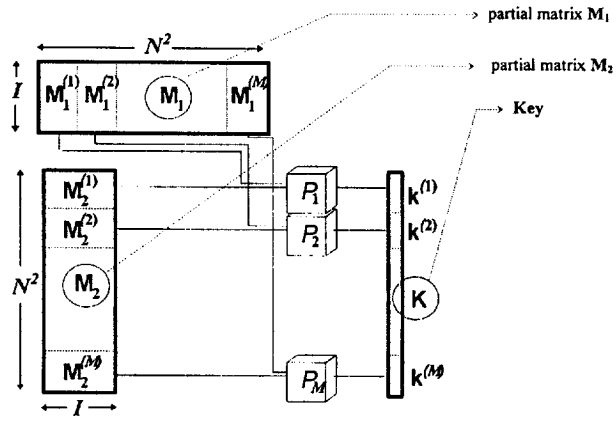

Fig. 3. The slice-oriented data mapping.

distribution of matrix multiplications among processors. A hierarchical architecture of processors reflects the structural choice already adopted for previous research on parallel implementation of associative memories with computation-intensive modelling [11].

The uniform distribution of the computational load is attained by a suitable data-allocation strategy, which consists in splitting all data matrixes (images, memory, etc.) into as many *slices* as the number of processors (Fig. 3). Therefore, each network node performs local computations on the assigned portions of the data sets. This basic architecture was also adopted in the above-mentioned implementation of an image classifier using noise-like coding associative memories [11]; therefore, the present research also allows a comparative evaluation of different approaches to the same problem.

At the initialization step, the ROOT (master) processor sends to each mth slave the assigned portions, $\mathbf{M}_1^{(m)}$ and $\mathbf{M}_2^{(m)}$, of $\mathbf{M}_1$ and $\mathbf{M}_2$, respectively. For simplicity and without loss of generality, we assume that all slaves ($m = 1, \ldots, M$) receive portions of equal size. Uniform splitting is also valid for the key pattern, $\mathbf{K}$, which is divided into m slices for on-line recall. As a result, the sizes of the data sets $\mathbf{M}_1^{(m)}$, $\mathbf{M}_2^{(m)}$ and $\boldsymbol{k}^{(m)}$ on each processor are given by $[I, N^2/M]$, $[N^2/M, I]$, and $[N^2/M]$, respectively. The initialization step is executed *offline* and is not considered in the performance analysis of the system.

Run-time memory recall involves the following steps:

(1) the ROOT node splits the input image, $\mathbf{K}$, into as many portions as the available processors, and downloads the resulting 'slices', $\boldsymbol{k}^{(m)}$, to slave nodes accordingly;

(2) each slave computes the local portion of the first intermediate product, $\boldsymbol{v}^{(m)} = \mathbf{M}_1^{(m)}\boldsymbol{k}^{(m)}$;

(3) vectors $\boldsymbol{v}^{(m)}$ (of size $[I]$) are recollected at the ROOT level; their vector sum yields the intermediate vector, $\boldsymbol{v}$;

(4) $\boldsymbol{v}$ is broadcast to all slaves;

(5) each slave computes the local portion of the actual recall: $\boldsymbol{r}^{(m)} = \mathbf{M}_2^{(m)}\boldsymbol{v}$;

(6) partial recall vectors are reassembled at the ROOT, which builds up the final result, r.

In the above algorithm, inter-node communications take place at steps (1), (3), (4), and (6). It is worth stressing, however, that these phases involve at most vector quantities; hence they play a minor role for the eventual efficiency. Conversely, the specific data-allocation strategy manages to keep the bulk of the computation-intensive process at the local level, thus increasing the overall efficiency.

3.2. Theoretical analysis of the system's efficiency

This section considers the algorithm previously outlined, and derives an analytical expression for the structure efficiency. Such an expression makes it possible to predict the overall system's performance. The following notations and conventions will be used for the analysis:

- τ_{pixel} = time required to transmit a pixel through a physical link between two processors;
- τ_{mul} = time to perform a floating-point multiplication;
- τ_{sum} = time to perform a floating-point sum;
- L = total network depth equal to the number of hierarchy levels supported by the processor tree (0 = root level, ..., L);
- M = total number of processors.

3.2.1. Communication overhead

Link communications among transputers proceed sequentially on each node, because of each processor's inability to support a parallel memory access. This limitation has major consequences on the eventual performance, as the time for information broadcasting from each node to lower ones is given by the sum of all individual transmission phases. Run-time operation involves different types of communications:

- top-down transmission of key slices, $\boldsymbol{k}^{(m)}$, $T_c^{(k)}$;
- bottom-up collection of partial vectors $\boldsymbol{v}^{(m)}$, $T_c^{(v\uparrow)}$;
- top-down broadcast of the intermediate vector $\boldsymbol{v}$, $T_c^{(v\downarrow)}$;
- bottom-up collection of portions of vector $\boldsymbol{r}^{(m)}$, $T_c^{(r)}$.

Data transmission affects communication costs selectively, depending on whether common information is broadcast or individual information is sent to each processor. Two structural factors can be defined accordingly.

When common information must be broadcast to all slaves, the cost term, $F^{(b)}$, is expressed as:

$$F^{(b)} = \sum_{l=0}^{L} \max_{j} \{BF(l, j)\} \tag{9}$$

where $BF(l, j)$ indicates the number of outgoing downward links of processor (l, j) (branching factor of the jth processor at level l). Summing the highest branching factor at each level lets the total broadcast time depend on the largest subtree in the network. Thus the term $F^{(b)}$ gives a structural and general measure of the topology effect on

broadcast communications; in fact, the amount of transmitted data is not proportional to the number of network nodes.

When transmitted information differs from one node to another (in other words, when the path differs), the structural cost, $F^{(d)}$, is expressed as:

$$F^{(d)} = \sum_{l=0}^{L} \max_{j} \left\{ \sum_{h=1}^{BF(l,j)} D(l, j, h) \right\} \tag{10}$$

where $D(l, j, h)$ is the size (number of nodes) of the subtree spanned from the h^{th} link of the j^{th} processor at level l. This quantity is architecture-dependent, and can be computed once for ever when the global processor hierarchy is defined. The function $D(\ )$ was previously used in Ref. [11] to characterize hierarchical topologies, and returns a communication cost proportional to the total weight (number of nodes) of the specific subtree spanned from a given link. For example, for the architecture shown in Fig. 1, $D(l, j, h)$ has the following values:

$$\begin{aligned} &\text{level } 0, \quad D(0, j, h) = 4, \quad \forall j, h \\ &\text{level } 1, \quad D(1, j, h) = 1, \quad \forall j, h \\ &\text{level } 2, \quad D(3, j, h) = 0, \quad \forall j, h \end{aligned}$$

As opposed to $F^{(b)}$, which takes into account the largest subtree in the network but can exploit some parallelism, the term $F^{(d)}$ yields a structural cost proportional to the actual network weight (where the amount of transmitted data is proportional to the number of nodes). $F^{(d)}$ is structure-dependent and can be computed once when defining the global processor hierarchy.

With these notations, communication costs can be expressed as follows:

$$T_c^{(k)} = T_c^{(r)} = \frac{N^2}{M} F^{(d)} \tau_{\text{pixel}} \tag{11}$$

$$T_c^{(v\uparrow)} = IF^{(d)} \tau_{\text{pixel}} \tag{12}$$

$$T_c^{(v\downarrow)} = IF^{(b)} \tau_{\text{pixel}} \tag{13}$$

3.2.2. Computational timings

At run time, each processor performs two different computations at the local level:
- time to compute the intermediate product $\boldsymbol{v}^{(m)}$, $T_p^{(v)}$;
- time to compute the local recall $\boldsymbol{r}^{(m)}$, $T_p^{(r)}$.

Finally, the time, $T_p^{(\Sigma)}$, for summing the M intermediate vectors on the ROOT node at step (3) must also be taken into account. The timings associated with such processes, considering the matrix-vector multiplications at steps (2) and (5), are given by:

$$T_p^{(v)} = I \left[\frac{N^2}{M} \tau_{\text{mul}} + \left(\frac{N^2}{M} - 1 \right) \tau_{\text{sum}} \right] \tag{14}$$

$$T_p^{(r)} = \frac{N^2}{M}\left[I\tau_{mul} + (I-1)\tau_{sum}\right] \tag{15}$$

$$T_p^{(\Sigma)} = I(M-1)\tau_{sum} \tag{16}$$

3.2.3. Analysis of the architecture efficiency

The parallel system's efficiency is defined as $\eta = T_{seq}/(MT_{par})$, where T_{seq} and T_{par} denote the timings for the sequential execution of the recall process on one transputer and on the parallel architecture, respectively. The former includes the timings for two matrix multiplications:

$$\begin{aligned} T_{seq} &= I\left[N^2\tau_{mul} + (N^2-1)\tau_{sum}\right] + N^2\left[I\tau_{mul} + (I-1)\tau_{sum}\right] \\ &= 2IN^2\tau_{mul} + (2IN^2 - I - N^2)\tau_{sum} \end{aligned} \tag{17}$$

The timing of the parallel execution is obtained by combining Eqs. (11)–(13) and Eqs. (14)–(16):

$$\begin{aligned} T_{par} &= T_p^{(v)} + T_p^{(r)} + T_p^{(\Sigma)} + T_c^{(k)} + T_c^{(r)} + T_c^{(v\uparrow)} + T_c^{(v\downarrow)} \\ &= \frac{2IN^2}{M}\tau_{mul} + (2I-1)\frac{N^2}{M}\tau_{sum} + I(M-2)\tau_{sum} + \left[\left(2\frac{N^2}{M} + I\right)F^{(d)} \right. \\ &\qquad \left. + IF^{(b)}\right]\tau_{pixel} \end{aligned} \tag{18}$$

The overall system efficiency can then be derived from Eqs. (17) and (18):

$$\begin{aligned} \eta &= \frac{1}{M}\cdot\frac{2IN^2\tau_{mul} + (2IN^2 - I - N^2)\tau_{sum}}{(2IN^2/M)\tau_{mul} + (2I-1)(N^2/M)\tau_{sum} + I(M-2)\tau_{sum} + \left[\left(2(N^2/M) + I\right)F^{(d)} + IF^{(b)}\right]\tau_{pixel}} \\ &= \left(1 - \frac{(I+N^2)\tau_{sum}}{2IN^2(\tau_{mul}+\tau_{sum})}\right)\Bigg/\left(1 + \frac{[IM(M-2) - N^2]\tau_{sum} + [(2N^2+IM)F^{(d)} + IMF^{(b)}]\tau_{pixel}}{2IN^2(\tau_{mul}+\tau_{sum})}\right) \end{aligned} \tag{19}$$

The efficiency expression makes it possible to verify consistency, i.e., that:

$$\begin{aligned} \eta > 1 \;&\Leftrightarrow\; -(I+N^2)\tau_{sum} \\ &\qquad < \left[IM(M-2) - N^2\right]\tau_{sum} + \left[(2N^2+IM)F^{(d)} + IMF^{(b)}\right]\tau_{pixel} \\ &\Leftrightarrow 0 < (M^2 - 2M + 1) + \frac{\tau_{pixel}}{I\tau_{sum}}\left[(2N^2+IM)F^{(d)} + IMF^{(b)}\right] \\ &\Leftrightarrow \begin{cases} M^2 - 2M + 1 > 0 & \text{True } \forall M > 1 \\ \dfrac{\tau_{pixel}}{I\tau_{sum}}\left[(2N^2+IM)F^{(d)} + IMF^{(b)}\right] > 0 & \text{Always true} \end{cases} \end{aligned}$$

Eq. (19) also shows that, in a single-processor architecture ($M = 1$), efficiency evaluates $\eta = 1$, as communication costs Eqs. (9) and (10) nullify.

At present, performances of computers permit to perform floating-point operations (sum and multiplication) by comparable times. For this reason, Eq. (19) can be simplified as follows:

$$\eta = \frac{(4I-1)N^2 - I}{(4I-1)N^2 + (M-2)IM + \alpha\left[(2N^2+M)F^{(d)} + IMF^{(b)}\right]} \cong \frac{4IN^2}{4IN^2 + (M-2)IM + \alpha\left[(2N^2+M)F^{(d)} + IMF^{(b)}\right]}$$

$$\text{where } \alpha = \frac{\tau_{\text{pixel}}}{\tau} \text{ and } \tau = \tau_{\text{sum}} = \tau_{\text{mul}}. \tag{20}$$

4. Experimental results

Experiments aimed at assessing the system's performance in terms of both computational efficiency and classification accuracy. For the HW support, we used an evaluation system including 8 transputers of the T800 family, arranged as shown in Fig. 2. The visual testbed included a prototype set of binary images drawn from different domains, including cell nuclei, human faces, and geometrical shapes. This data set involved a difficult problem because pictures belonging to the same domain (e.g., faces or cell nuclei) were quite similar to one another, and formed thick clusters difficult to separate. The image size was always set to $N^2 = 128 \times 128$ pixels.

4.1. Efficiency measurements

Another important verification of the consistency of the research presented in this paper concerned the correctness of the predicted efficiency (Eq. (19)) as compared with empirical evidence. In this sense, the experimental setup we used for evaluation involved a board with $M = 8$ processors of the T800 family, using inter-transputer links operating at 10 Mbit/s, giving $\tau_{\text{pixel}} = 4.48$ μs; the time required for a floating-point operation amounted to $\tau = 4.32$ μs. The hierarchical topology illustrated in Fig. 2 and the definition of the structural cost functions leads to the communication factors $F^{(b)} = 5$ and $F^{(d)} = 11$ as follows:

$$F^{(b)} = \sum_{l=0}^{2} \max_{j}\{BF(l, j)\} = 2 + 3 + 0 = 5;$$

$$F^{(d)} = \sum_{l=0}^{2} \max_{j}\left\{\sum_{h=1}^{BF(l,j)} D(l, j, h)\right\} = (4+4) + (1+1+1) + 0 = 11.$$

The network symmetry makes the max( ) operators irrelevant.

Using these technical values in Eq. (19) yields efficiency estimates for the specific architecture. Table 2 allows a comparison between predicted and measured values; the

Table 2
Efficiency and speedup results

	T_{seq}	T_{par}	Measured η	Predicted η	Speedup
$I = 8$	1.777 s	0.411 s	0.54	0.583	4.666
$I = 16$	3.488 s	0.586 s	0.74	0.736	5.891

T_{seq} = recall timings for the sequential algorithm.
T_{par} = recall timings for the parallel algorithm.

fit between experimental and expected values demonstrates the validity of the theoretical model.

From a strictly technical point of view, computational timings appear quite satisfactory (a 16-class classifier operates in about half a second) and, at the same time, the values of the overall efficiency are quite significant. These results refer to the specific mathematical model described (mainly based on matrix algebra). However, timing performances can be further enhanced by using 20 Mbit/s links and including unrolling techniques in the floating-point vector computations.

As to the overall classification performance, some interesting issues stem from a comparison of a DAM-based classifier with the schema described in Ref. [11] and using noise-like coding memories. In this respect, Kohonen's model exhibits a greater robustness, especially against speckle noise, whereas insensitivity to Gaussian noise is almost equivalent for the two models. On the other hand, it is worth stressing that DAMs require (even in the reformulated version) a much larger memory occupation than noise-like coding memories to store the same amount of information. (Incidentally, we recall that the size of a noise-like coding memory is constant and equal to $2N^2$.) As a consequence, the increased insensitivity of a DAM-based classifier is obtained at the cost of heavier hardware requirements, hence the ultimate choice will be driven by a performance/cost tradeoff dependent on the actual application domain.

This line of research ultimately confirms the interest in using associative memory models as support devices to enhance a classification system's performance in a flexible and effective manner, by providing trainable and adaptive components whose relatively inexpensive parallel implementation makes real-time application feasible.

5. Conclusions

The paper has addressed a basic model of associative memory and described its use for high-dimensional image recognition.

The presented research also exhibits novel aspects from a theoretical perspective. The model reformulation makes a practical implementation feasible without affecting the memory performance; at the same time, the architectural choice and the implementation strategy have defined the method's effectiveness quantitatively, and proved its notable scalable properties.

The inclusion of DAMs inside an image-processing system appears very promising, especially when considering the possibility of adopting hetero-associative schemata, which have proved effective in other related approaches presented in the literature.

Acknowledgements

The authors gratefully acknowledge an anonymous reviewer, whose careful reading and constructive suggestions greatly contributed to increase the paper quality and soundness.

References

[1] D. Anguita, G. Parodi, R. Zunino, Associative structures for vision, Multidimensional Syst. Signal Process. 5 (1994) 75–96.
[2] V. Bharadwaj, D. Ghose, V. Mani, Optimal sequencing and arrangement in distributed single-level tree networks with communication delays, IEEE Trans. Parallel Distr. Syst. 5 (9) (1994) 968–976.
[3] M.Y. Chan, S.J. Lee, Fault-tolerant embedding of complete binary trees in hypercubes, IEEE Trans. Parallel Distr. Syst. 4 (3) (1993) 277–288.
[4] A. Diaspro, G. Parodi, R. Zunino, A performance analysis of an associative system for image classification, Pattern Recognition Lett. 14 (11) (1993) 861–868.
[5] J.F. Jenq, S. Sahni, Image shrinking and expanding on a pyramid, IEEE Trans. Parallel Distr. Syst. 4 (11) (1993) 1291–1296.
[6] T. Kohonen, Correlation matrix memories, IEEE Trans. Comput. C-21 (1972) 353–359.
[7] T. Kohonen, E. Oja, P. Lehtio, Storage and processing of information in DAM systems, in: G.E. Hinton, J.A. Anderson (Eds.), Parallel Models of Associative Memory - Updated ed., Lawrence Erlbaum, New York, 1989.
[8] Q.M. Malluhi, M.A. Bayoumi, The hierarchical hypercube: A new interconnection topology for massively parallel systems, IEEE Trans. Parallel Distr. Syst. 5 (1) (1994) 17–30.
[9] C. Moneta, G. Vernazza, R. Zunino, On the need for integrated approaches to image understanding, Eur. Trans. Telecommun. 3 (5) (1992) 41–54.
[10] R.C. Nelson, Visual homing using an associative memory, Proc. Image Understanding Workshop 1989, Morgan Kaufman, Los Altos CA, pp. 245–262.
[11] F. Pagano, G. Parodi, R. Zunino, Parallel implementation of associative memories for image classification, Parallel Comput. 19 (6) (1993) 667–684.
[12] W. Polzleitner, H. Wechsler, Selective and focused invariant recognition using distributed associative memories (DAMs), IEEE Trans. Pattern Anal. Machine Intell. 12 (8) (1990) 809–814.
[13] S.L. Scott, G.S. Sohi, The use of feedback in multiprocessors and its application to tree saturation control, IEEE Trans. Parallel Distr. Syst. 1 (4) (1990) 385–398.
[14] H.Y. Youn, A.D. Singh, On implementing large binary tree architectures in VLSI and WSI, IEEE Trans. Comput. 38 (4) (1989) 526–537.
[15] H. Wechsler, J.L. Zimmermann, 2-D invariant object recognition using distributed associative memory, IEEE Trans. Pattern Anal. Machine Intell. 10 (6) (1988) 811–821.