LBGS: A Smart Approach for Very Large Data Sets Vector Quantization

Giuseppe Campobello , Mirko Mantineo , Giuseppe Patanè and Marco Russo

Abstract

In this paper, LBGS, a new parallel/distributed technique for Vector Quantization is presented. It derives from the well known LBG algorithm and has been designed for very complex problems where both large data sets and large codebooks are involved. Several heuristics have been introduced to make it suitable for implementation on parallel/distributed hardware. These lead to a slight deterioration of the quantization error with respect to the serial version but a large improvement in computing efficiency.

Keywords

Clustering, Vector Quantization, Unsupervised Learning, parallel, distributed, learning

I. Introduction

Often, both in the industrial and the scientific world, clustering and vector quantization (VQ) techniques are applied to very complex problems. The fields of application of such techniques range from texture and image segmentation [1,2], magnetic resonance imaging [3] and computer vision [4] to information retrieval [5], machine learning [6] etc. When the task is very complex, i.e. both large data sets and large codebooks are involved (for example, in image segmentation [7], image coding [8,9] and speech coding [10,11]), the computing time required by a classical approach [12,13] may be prohibitive. A possible solution to this problem is the use of parallel and distributed computing systems [14–20]. Recent studies [21,22] have shown that, for an efficient parallel implementation of the most advanced (serial) VQ techniques, it is necessary to use computing systems whose hardware provides a large bandwidth for inter-process communications. Only in this way, is it possible to use a large number of Processing Elements (PEs) efficiently to obtain a meaningful decrease in the computing time. However, hardware systems with the required communication bandwidth are very expensive and, for this reason, they are not within everybody's reach. So, in recent years, low-cost distributed computing systems, realized by clusters of personal computers (farms of PCs) have increased a lot in popularity.

Sometimes, VQ problems are so complex that, even though powerful hardware is used, the computing time is too long or completely unacceptable. In these circumstances, the employment of particular VQ techniques can be useful that, at the cost of a deterioration in the quality of the final results, perform a considerably lower number of calculations with respect to traditional algorithms.

The work presented in this paper, LBGS, belongs to this field of research. In particular, it proposes a smart technique that obtains very good results particularly with the most complex problems, i.e. when both the cardinality of the data set and the codebook are considerable [7–11]. Its name derives from the LBG algorithm [12] which has been widely modified to make it more suitable for implementation on parallel and distributed systems. In particular, the final "S" has been added to *LBG* to specify *Super-clustering*, one of the new concepts we have introduced and that will be detailed in Sections VI-VII.

The starting point of this work was the idea of developing a new parallel algorithm with a minimized inter-process communication and with the tasks involved in the quantization process working as independently as possible from each other. Besides, we tried to reduce the number of calculations performed by accepting, as regards the final quantization error, lower-quality results than a classic serial approach. Moreover, the user can choose, to some extent, a compromise between precision and speed. However, we wish to underline that the main objective of the work is to favour as far as possible the error, i.e. the algorithm presented tries to obtain almost the same error as the original serial technique. On the other hand, in literature, we can find works favouring speed at the expense of error [23]. In our opinion, the objective has been reached and the results are relevant.

The particular features that we have briefly described make LBGS very suitable for a parallel implementation on a cluster of PCs. As we said before, this is a low-cost distributed computing system whose main limitation is the low availability of bandwidth for inter-process communication. This does not mean that LBGS is an algorithm for a cluster

Giuseppe Campobello is with the Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, ITALY ; e-mail: gcampo@ai.unime.it

Mirko Mantineo is with the Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, ITALY

Giuseppe Patanè is with the MPG - STMicroelectronics, Stradale Primosole, 50, 95121 Catania, ITALY; e-mail: giuseppe.patane@st.com

Marco Russo (corresponding author) is with the Department of Physics and Astronomy, Univeristy of Catania and National Institute of Nuclear Physics (INFN) Section of Catania, Viale A. Doria, 6 , 95125 Catania - ITALY; e-mail: marco.russo@ct.infn.it

of PCs because the techniques developed for LBGS have general validity and, when more powerful computing systems are available, it is possible to obtain lower computing times or, in an equal time, lower errors.

This paper is organized as follows: in Section II the computing system used for the implementation of LBGS is presented; Section III is a brief summary of the terminology used; Section IV presents the basics of VQ; Section V is an overview and a classification of several algorithms existing in literature that can be used in the same application domain as LBGS; in Section VI, LBGS is presented; in Section VII, a more detailed description of LBGS is given; in Section VIII, the results obtained by LBGS are presented and discussed; Section IX reports the authors' conclusions.

II. The MULTISOFT Machine

The elaboration system used for the implementation of LBGS is called the MULTISOFT Machine (see Fig. 1). It is a cluster of PCs with the LINUX operating system and consists of 32 Processing Elements (PEs) and a server (duplicated for redundancy) acting also as a gateway towards the external LAN. All of the 32 PEs are mono-processor machines, 7 are based on Intel Pentium-II 233 and 25 on Intel Celeron 366. All of them are endowed with 32 MB of RAM, one hard disk and one Fast-Ethernet network card. One server is a bi-processor computer with two Intel Pentium-II 300s and two network cards.

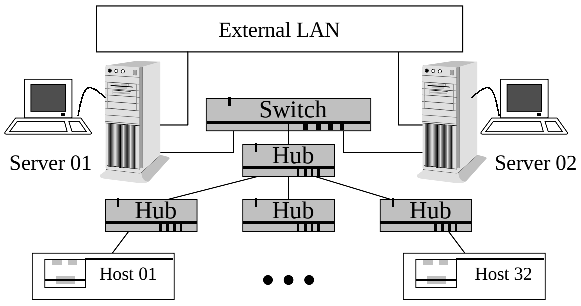

Fig. 1. The MULTISOFT Machine

The PEs are interconnected by a dedicated Fast Ethernet network and all of them share the same bus. Physical connections have been realized by means of four 12-port hubs distributed on two levels as shown in Fig. 1. The highest level hub is connected to a switch, where the servers are directly connected, too.

Inter-process communication is realized with the Parallel Virtual Machine (PVM) software[1]. It gives message passing facilities and makes the user see several computers as a large and single virtual machine where each process (task, according to the PVM nomenclature) has a unique identifier (task identifier, tid) inside the virtual machine. PVM uses the transport functionality provided by the TCP/(UDP/)IP that are directly implemented inside the kernel of LINUX.

For diverse reasons, at the moment of the tests, some PEs were unavailable, So, in this paper, we will show our results using only up to 20 PEs, all Celeron 366-equipped.

The MULTISOFT Machine has also been used for implementing other algorithms cited throughout the paper such as our implementation of PKM, PARELBG [22] and PAUL [24].

III. Brief discussion of the terminology employed

Now, let us introduce some terms and acronyms that we will use in the remainder of the paper and that will be useful to make the explanation of the concepts easier.

A. Performance and efficiency

The algorithms we deal with will be evaluated from the point of view of both the final quantization error and computational efficiency. For the sake of brevity, in the former case, we will use simply the term **performance**, in the latter we will use **efficiency**.

The term efficiency is for us of qualitative nature. We say that, given a precise hardware, an algorithm is efficient if the possibility exists of implementing it with a software running on this hardware that uses the resources at its disposal well. So, the efficiency is closely related to the hardware. Then one algorithm can be efficient for a particular computer architecture and not efficient for another architecture.

For example, in the particular case of parallel and/or distributed architectures, we can find several algorithms that are efficient when only one of the nodes available is used (we call this serial case) and absolutely not efficient when more nodes are used. Clearly, in this particular domain the efficiency is linked to the speedup: an efficient algorithm for parallel architectures reaches good speedups otherwise it is not efficient. This is mainly due to the intrinsically serial

[1]It is available for free at http://www.netlib.org/pvm

nature of some of the operations executed (Amdhal's law [25]), the overhead introduced by inter-process communication and the synchronization between tasks required at each iteration. They can constitute a decided limitation to the achievable speed-up, also when powerful computing systems are available.

B. Parallel algorithms

Often, the parallel implementation of an efficient serial clustering algorithm is not very efficient.

A possible improvement is the substitution of intrinsically serial operations with efficiently parallelizable versions that try to approximate the original ones as well as possible. The cost of such an increase in efficiency is, generally, a decrease in performance.

According to these considerations, it is useful, to simplify the treatment in the remainder of the paper, to classify the parallel VQ we will deal with into two categories:

- *Perfect Implementation of a Serial Algorithm (PISA).* These are the algorithms that make, in parallel, the same operations as the serial reference version obtaining, of course, the same identical performances. PISA VQ techniques are often not efficient. The reasons for such behaviour will be analysed in detail in Section V-C;
- *Modified Implementation of a Serial Algorithm (MISA).* These are the algorithms that have a serial algorithm as a reference but intrinsically serial operations are substituted by efficiently parallelizable versions that try to approximate the original ones as well as possible. Generally, they are more efficient than PISA algorithms but their performances are worse.

C. Table of the main symbols used in the paper

In table I we report the main symbols used in the paper with the aim of increasing the readability of the paper.

Symbol	Description
N_C	total number of codewords
N_P	total number of patterns in the input data-set
MQE	Mean Quantization Error
D_{curr}	MQE at the current iteration
D_{prev}	MQE at the previous iteration
MSE	MQE when the Squared Error (SE) is used as a distortion measure
RMSE	square root of the MSE
ϵ	threshold for the termination of the algorithgm
N	number of slave tasks
N_{C_i}	number of codewords assigned to the ith task
N_{SC}	total number of super-codewords
N_{ST}	mean number of super-codewords per task ($\frac{N_{SC}}{N}$)
n_s	number of iterations to be performed before a new super-clustering of the codewords occurs
n_c	number of iterations to be performed before the slave compares its codewords again with the whole super-codebook
S	speed-up

TABLE I
SYMBOLS

IV. Vector Quantization

A. Definition

The objective of VQ is the representation of a set of vectors $\mathbf{x} \in X \subseteq \Re^k$ by a set, $Y = \{\mathbf{y}_1, ..., \mathbf{y}_{N_C}\}$, of N_C reference vectors in $\Re^k$. Y is called *codebook* and its elements *codewords*. The vectors of X are called also *input patterns* or *input vectors*. So, a VQ can be represented as a function: $q : X \longrightarrow Y$. The knowledge of q permits us to obtain a partition $\mathcal{S}$ of X constituted by the N_C subsets S_i (called cells):

$$S_i = \{\mathbf{x} \in X : q(\mathbf{x}) = \mathbf{y}_i\} \quad i = 1, \ldots, N_C \tag{1}$$

B. Quantization Error (QE).

The QE is the value assumed by $d(\mathbf{x}, q(\mathbf{x}))$, where d is a generic distance operator for vectors. Several functions can be adopted as distortion measures [12]; in this paper we will consider the Square Error (SE), whose formulation is

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} (x_i - y_i)^2 \tag{2}$$

The Mean QE (MQE) is used to evaluate the performance of a quantizer. In particular, if the SE is adopted for the distortion measure, the MQE is called Mean Square Error (MSE). We will also use the square root of the MSE (RMSE).

In general terms, when X is constituted by a finite number (N_P) of elements, the MQE is given by:

$$\mathrm{MQE} \equiv D(\{Y, \mathcal{S}\}) = \frac{1}{N_P} \sum_{i=1}^{N_C} D_i \tag{3}$$

where we indicate with D_i the ith cell total distortion:

$$D_i = \sum_{n:\mathbf{x}_n \in S_i} d(\mathbf{x}_n, \mathbf{y}_i) \tag{4}$$

Equations (3) and (4) show that the MQE can be calculated as a function (D) of the codebook (Y) and the partition ($\mathcal{S}$).

Now, we report two important conditions we will use throughout this paper that are necessary for calculating the optimal partition (when the codebook is fixed) and the optimal codebook (when the partition is fixed) [12].

• **Nearest Neighbor Condition (NNC).** Given a fixed codebook Y, the NNC consists in assigning the nearest codeword to each input vector. So, it is possible to divide the input data set in the following manner:

$$\begin{aligned} \bar{S}_i \quad = \quad & \{\mathbf{x} \in X : \; d(\mathbf{x}, \mathbf{y}_i) \le d(\mathbf{x}, \mathbf{y}_j), \\ & j = 1, ..., N_C, \; j \neq i\} \quad i = 1, ..., N_C \end{aligned} \tag{5}$$

The sets $\bar{S}_i$ just defined, constitute a partition of the input data set. This is called the "Voronoi Partition" [26] and is referred to by the symbol $\mathcal{P}(Y) = \{\bar{S}_1, \cdots, \bar{S}_{N_C}\}$. It is possible to demonstrate that the Voronoi partition is optimal for that codebook [12].

• **Centroid Condition (CC).** Given a fixed partition $\mathcal{S}$, the CC concerns the procedure for finding the optimal codebook. This is the codebook constituted by the centroid of each cell [12].

If we consider the set $A \subset \Re^k$ constituted by N_A elements and the SE is adopted as the measure for distances, its centroid $\bar{\mathbf{x}}(A)$ is:

$$\bar{\mathbf{x}}(A) = \frac{1}{N_A} \sum_{\mathbf{x} \in A} \mathbf{x} \tag{6}$$

The codebook $\bar{X}(\mathcal{S})$ constituted by the centroid of all the cells of $\mathcal{S}$:

$$\bar{X}(\mathcal{S}) \equiv \{\bar{\mathbf{x}}(S_i); \; i = 1, ..., N_C\} \tag{7}$$

is optimum for that partition [12].

V. Previous works

In this section, we will present a brief overview of some VQ techniques, both serial (LBG [12], K-means [27] and ELBG [13]) and parallel (PKM [14–19, 21, 28], PARELBG [21, 22], P-CLUSTER [20, 29, 30] and PAUL [24]), selected from the large existing literature.

A. Serial approach

A.1 LBG and K-means

LBG [12] is an iterative algorithm, that, N_C being fixed, iteration by iteration, produces a quantizer better than or equal to the one at the previous iteration. It is analogous, practically equivalent [31], to the traditional K-means [27], another algorithm very well known in literature. The steps through which LBG develops can be summarized as follows:
1. **initialization**. This phase consists of the choice of an initial codebook. Several techniques are reported in literature on the matter; among them, we mention the random initialization and the initialization by splitting [12];
2. **partition calculation**. Given the current codebook, the related Voronoi partition is calculated according to the NNC (5);
3. **termination condition check**. The MQE at the current iteration (D_{curr}) is compared with the one at the previous iteration (D_{prev}). If the ratio $|D_{\text{prev}} - D_{\text{curr}}|/D_{\text{prev}}$ is less than a prefixed threshold (ϵ) then the algorithm ends; otherwise, it continues with the next step;
4. **new codebook calculation**. Given the current partition, the new codebook is calculated according to the CC (7);
5. **return to step 2**.

A.2 Discussion regarding LBG

The main drawback of LBG is its high sensitivity to the choice of the initial codebook. This means that, if the initialization is not good, it converges towards a bad local minimum because a certain quota of codewords remain unused (i.e. they generate empty cells) or badly positioned; the consequence is a deterioration in performances.

Regarding the problem of the unused codewords, some simple solutions were proposed by the same authors of LBG. For example, such codewords could be removed and, at the same time, the cells with the largest distortions (according to (4)) split.

However, there is a much more complicated problem which concerns the codewords that are badly positioned, although they do not generate empty cells [13]. To try to solve this, several algorithms have been proposed in literature [13,32–38].

A.3 ELBG

An interesting algorithm among those mentioned above is *The Enhanced LBG* (ELBG) [13, 32, 33]. Its analysis is important for two reasons:
1. for each of the tests performed, ELBG obtained performances better than or equal to the other algorithms considered; so, it is a valid reference for evaluating other techniques;
2. a parallel version of ELBG [22], briefly described below, exists and can be used for comparison.

ELBG derives directly from LBG by adding a new computational step, called the *ELBG-block*, to manage the shifting of badly positioned codewords.

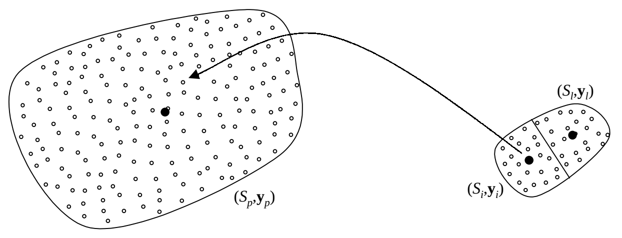

Fig. 2. ELBG: A situation where the codewords are badly distributed, but locally optimally placed. The arrow indicates a possible shift that, after a rearrangement, gives lower error

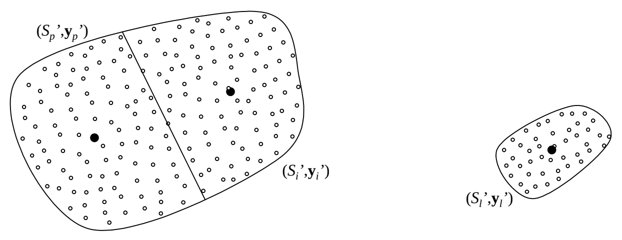

Fig. 3. ELBG: A better distribution for the codewords than the one shown in Fig. 2

In brief, the basic idea on which ELBG is based is described below.

The aim of ELBG is to escape from the local minima by attempting to equalize the distortions introduced by the cells (D_i). In Fig. 2 a typical example of a local minimum is reported; the arrow indicates a possible shift that, after a rearrangement, gives lower error. In Fig. 3 a better distribution of the codewords, that arises by executing the shift indicated by the arrow and some local adjustments, is shown. The ELBG-block attempts to remedy the situation of local minimum by smartly performing several shifts such as the one illustrated in Figs. 2 and 3.

The shift represented in Figs. 2 and 3 equalizes the total distortions (D_i (eq. 4)) of the three represented cells. The mathematical justification of such an operation was given in [13,32]. The basic idea of the ELBG-block is to go towards the desired equalization by joining a low-distortion cell with a cell adjacent to it. At the same time, a high-distortion cell is split into two smaller ones. So, it is as if the low-distortion codeword is moved inside the high-distortion cell. This is a Shift of Codeword Attempt (SoCA). If it produces a decrease in the MQE, then the SoCA is confirmed, i.e. a Shift of Codeword (SoC) is executed. Otherwise, the shift is discarded. Several SoCAs are executed inside the ELBG-block. According to these definitions, Figs. 2 and 3 represent a simple example of a SoCA.

The execution of the ELBG-block introduces an overhead that has been kept low (around 10-20%) thanks to particular sub-optimum techniques for local rearrangements and a data structure developed *ad hoc* [33]. Such are the benefits of the ELBG-block that ELBG is, practically, independent of the choice of the initial codebook, has performances better than or equal to the other algorithms used for comparisons and converges after a low number of iterations.

B. *Distributed approach*

In this sub-section we will describe several approaches, already existing in literature, to the realization of VQ algorithms on parallel and distributed architectures and they will be valued in terms of *performance* and *efficiency.* The parallel algorithms that will be briefly presented belong both to the PISA (PKM, PARELBG, P-CLUSTER) and MISA (PAUL) families.

B.1 Parallel K-means (PKM): a family of PISA algorithms

In literature, a large number of parallel and distributed implementations of LBG and K-means exist. Various hardware architectures have been employed such as, for example: specialized architectures [28], massively parallel processors [14], transputers [15,16] and networks of workstations [17–19,21]. The idea at the basis of such techniques is the subdivision of the most time-consuming part of the algorithm (the calculation of the Voronoi partition) into a certain number of subtasks to be executed in parallel, while, the remaining operations (the calculation of the new centroids) are serially executed by a single process. For the sake of brevity, we will identify this kind of technique as Parallel K-means (PKM).

For a better explanation and quantification of the previous statements, we report a numeric example obtained by profiling the software that we developed to implement LBG. Let us consider the task of compressing the picture *giraffe*[2] (it is a 256 gray-levels image of 984×1488 pixels size, see Section VIII-B for details) with $N_C = 1024$. By profiling LBG, we can see that 98.7% of the time is spent in calculating the Voronoi partition; the remaining 1.3% is spent for I/O operations and the calculation of the new centroids. According to these numeric values, we can argue that the time spent in calculating the new centroids is negligible with respect to the remaining times; for this reason, in these algorithms, the new centroids are serially calculated by a single process.

In more detail, PKM uses a master-slave approach and a portion of the input patterns is assigned to each of the N slave tasks. The generic slave computes a quota of the global partition by comparing each input pattern in its portion with all of the codebook. At the end of the iteration, the master process collects the data related to the Voronoi partition from all of the slaves, calculates the new centroids and begins a new iteration with the broadcasting of the new codebook to the slaves. So, inter-process communication is rather low and tasks exchange information only at the beginning and the end of every iteration. PKM is a PISA algorithm; so, it gets the same final quantization error and number of iterations as the serial version. As regards efficiency, [21] shows that, for very complex problems, PKM achieves quite high speed-ups also when a large number of processors are involved.

B.2 Discussion regarding PKM techniques

As we said previously, the main drawback of LBG and K-means is their sensitivity to the choice of the initial codebook; of course, such a drawback is inherited by the related PISA algorithms. Regarding efficiency, let us, now, report a detailed analysis of the factors that can keep the speed-up lower than its maximum theoretical value, i.e. the linear speed-up. They are:

1. *execution of non-parallelizable operations.* In theory, the execution of intrinsically serial operations is a limitation to the maximum achievable speed-up. However, in this particular case, we have verified (also by numeric values) that the time required by the serial operations (essentially, the calculation of the new codebook) is negligible with respect to the time spent for the parallel operations (the calculation of the Voronoi partitions). So, as regards the efficiency of this algorithm, we can consider this problem to be less important than the following two;

[2]available at ftp://ai.unime.it/pub/data/giraffe.raw

2. *overhead introduced by inter-process communication.* The problem of the overhead introduced by inter-process communication is typical of parallel algorithms and, in general, it is strongly dependent on the hardware architecture employed. For example, with commodity super-computers, the communication system is realized by low-cost hardware, whereas, in traditional super-computers, more efficient, but much more expensive (several orders of magnitude) shared memory based solutions are available. In spite of the substantial money saving, the reduced performances of the communication systems can be a bottleneck when a large amount of data has to be transferred and/or a large number of processes are involved. In the particular case of the parallel hardware at our disposal, the MULTISOFT machine, in addition to the problems deriving from the poor communication system available, based on a Fast Ethernet network, we have to consider the problems deriving from the lack of a real broadcast function[3]. It would be very useful when, at the beginning of each iteration, the master sends the whole codebook to all of the slaves;
3. *synchronization between the tasks at each iteration.* A new iteration starts after the master computes the new codebook and distributes it to the slaves. However, to calculate the new codebook, the master needs, from each slave, the results related to the computing of its portion of the Voronoi partition. Practically, it is not possible to start a new iteration if all of the tasks have not completed the previous one and if all of the operations related to the transmission of the data computed by each slave have not ended.

B.3 Another PISA algorithm: PARELBG

PARELBG [21, 22] is the parallel implementation of ELBG that, as we have seen, is virtually insensitive to the choice of the initial codebook and is not affected by the problem of local minima as, on the contrary, K-means and PKM are. The approach followed for the implementation of ELBG is the same as PKM, i.e. a master-slave architecture where the slaves perform, in parallel, the computation of the Voronoi partition and the master computes the new codebook. Moreover, now, also the implementation of the ELBG-block is required. Unfortunately, the operations constituting the ELBG-block are intrinsically serial; so, its execution has to be entirely allocated to a single process (the master, in this case).

In practice, the master collects the results from the slaves (i.e. the computation of the Voronoi partition) and executes the ELBG-block. Then it calculates the new codebook.

The execution of the ELBG-block by a single process considerably reduces the performances of PARELBG with respect to PKM [21, 22]. In practice, while we could say that, in PKM, the time spent for the execution of serial operations was negligible with respect to the time spent in other operations, now it is not true any more. Besides, the same other limitations of PKM are present, i.e. the overhead introduced by inter-process communication and the synchronization between the tasks at each iteration.

B.4 A MISA algorithm: PAUL

PAUL (A Parallel Algorithm for Unsupervised Learning) [24] is a MISA algorithm deriving from ELBG. It has been designed for problems with a high number of input patterns and codewords in order to achieve performance very close to ELBG and PARELBG and improved efficiency with respect to the latter.

The most important change introduced in ELBG is the substitution of the ELBG-block with a new block that tries to emulate its functionalities but can be executed in parallel by all the tasks. In this way, the cost of serial operations is reduced with respect to PARELBG even though the problems deriving from the overhead of the inter-process communication and the synchronization between the tasks at each iteration remain.

The performances of PAUL are almost identical to those of PARELBG while, as regards efficiency, it is better than PARELBG.

B.5 P-CLUSTER and P-CLUSTER with algorithmic enhancements (PISA algorithms)

P-CLUSTER [20, 29, 30] is the parallel version of CLUSTER [34]. The latter is, in principle, similar to K-means with the addition of some steps whose aim is to escape from the situations of local minimum. However, its application domain consists of small data sets (fewer than 4000 patterns) and it performs well inside it.

So, with the aim of elaborating larger data sets, P-CLUSTER was developed [29]. In particular, P-CLUSTER was designed to be implemented on a Network of Workstations. In a second phase of their study, the authors developed several enhancements to their algorithm [20, 30] in order to prune as much computation as possible while preserving the clustering quality.

Nevertheless, even if all the pruning techniques proposed by its authors are applied to P-CLUSTER, the number of calculations it performs is rather high with respect to other, more efficient, algorithms [24].

[3]Although we use a network with broadcast functionalities, PVM implements such a function as a sequence of point-to-point transmissions to the slaves. From the point of view of the user interface, the implementation is totally transparent; but it is not the same as regards efficiency because the broadcast potentiality of the Fast Ethernet is not used. On the other hand, the management of a reliable broadcast communication is a rather complex problem [39].

Besides, when the number of the codewords increases, the number of computations explodes [20, 30], thus making its employment prohibitive for problems with a high number of codewords.

C. Considerations

The overview presented confirms that, practically, the problems to be solved for realizing high-efficiency algorithms are the same as we pointed out in V-B.2 for PKM techniques.

Now, let us summarize such issues and analyze the possible solutions for each of them. These considerations are the starting point of this work.

- *Serial operations.* When the incidence of these operations is not negligible, we can look for new ones that try to emulate the original and are suitable to be executed in parallel by the tasks. Practically, a PISA algorithm is turned into a MISA one. This is a compromise between performance and efficiency because the increase in the efficiency is, generally, paid for by a decrease in performances.
- *Overhead introduced by inter-process communication.* The only solution is the reduction of the information exchanged between the tasks to what is strictly necessary.
- *Synchronization between the tasks.* This problem can be sidestepped if the tasks are allowed to work in *asynchronous mode*, i.e. each task continues to communicate with the other tasks but, always, has operations to be executed and is never inactive waiting for somebody to send it data. Generally, this independence produces an increase in efficiency at the cost of a reduction in performances.

One last, very important consideration concerns all of the VQ techniques we have seen till now, both serial and parallel: large improvements in efficiency can be obtained by employing suitable methods for reducing the number of distance calculations performed during the Voronoi partition calculation (point 2 of LBG).

For example, the algorithm enhancements employed in P-CLUSTER use the mathematical properties of k-dimensional spaces and triangular inequality to allow the correct partitioning of data. They produce the same result as the traditional full-search technique, but with a lower number of calculations. The validity of such enhancements is general and the field of application is not restricted to P-CLUSTER.

Later in this paper, we will show the heuristics that we used for the reduction of calculations. Even though they do not guarantee the exact partitioning of data according to NNC (eq. (5)), they allow an increase in the efficiency of the parallel algorithm because they allow the tasks working in parallel to operate in asynchronous mode. Besides, we will see that the inevitable decrease in performance, deriving from the non-optimum operations performed, has been kept inside some limits thanks to the introduction of particular techniques that will be detailed in Sections VI-VII.

VI. General description of LBG Super-cluster (LBGS)

LBGS is a parallel MISA algorithm whose serial reference version is LBG. In LBGS, each process works with a portion of the input data set (IDS) and executes, autonomously, the traditional (serial) LBG for calculating a portion of the codebook. At the end, by joining all of the sub-codebooks, the global codebook is obtained.

This heuristic method allows the number of distance computations to be performed to be considerably reduced because, to calculate the Voronoi partition, each task compares only its portion of the IDS with its portion of codewords. Besides, since the tasks are independent of each other, their synchronization at the end of each iteration is not required any more. The combination of all these factors allows superlinear speedups to be obtained with respect to serial LBG.

In contrast to the benefits listed, the independence of the tasks is, however, a limitation; it can lead, as the final result, to (global) codebooks with quantization error greater than the classic serial LBG where, in calculating the Voronoi partition, each pattern is compared with all of the codewords.

In order to reduce the decrease in performance, we introduced a mechanism (the *super-clustering* of the codewords) for the exchange of information between the tasks; in this way, each of them has a global, even though approximate, vision of the whole codebook. The super-clustering is linked to another mechanism, the migration of patterns and codewords that, as we will see, allows the resolution of some bad partitioning cases deriving from the subdivision of both the IDS and the codebook.

A. The essential points of the new algorithm

Now, let us briefly analyze the main points of LBGS.

- *Master-slave architecture.* The whole algorithm is executed, in parallel, by one master and N slaves. From the point of view of computation, the master makes a negligible job: essentially, it is restricted to the initialization and termination operations of the algorithm. The slaves, on the contrary, perform all of the most important operations, i.e. the codebook and super-codebook calculation and the migration of the patterns and the codewords. Each slave, even though it continues to communicate with the master and the other slaves, works in a nearly autonomous way.
- *Complete division of data.* In LBGS, both the input patterns and the codewords are divided among the N slaves. Such an operation is performed by a *pre-clustering* of the whole IDS, thus obtaining the grouping of data by a spatial

neighborhood criterion. The *pre-clustering* is done by executing serial LBG (where $N_C = N$) with the N_P patterns of the IDS. In the end, N groups (called *macro-cells*) are obtained and each of them, represented by its codeword, is assigned to a task. Fig. 4 shows the situation after pre-clustering in the case of a simple bi-dimensional example that we will also use in the remainder of the paper.
The information extracted during the pre-clustering is also used to opportunely assign to each task the number of codewords (N_{C_i}) it has to calculate. So, it is possible to have N sub-quantizers evolving independently of each other. In other words, the ith task, to which the ith group of patterns, made up of N_{P_i} elements has been assigned, performs LBG with its patterns to obtain its portion of the codebook, made up of N_{C_i} codewords. By joining the N sub-codebooks calculated in this way, the global codebook, made up of N_C desired elements, is obtained. Fig. 5 reports the situation we have described, in the case of the simple bi-dimensional example we are considering.

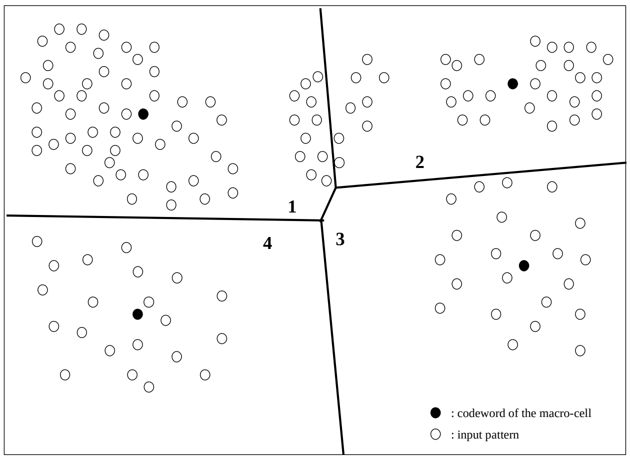

Fig. 4. Data pre-clustering. $N = 4$ macro-cells are reported, each represented by its codeword

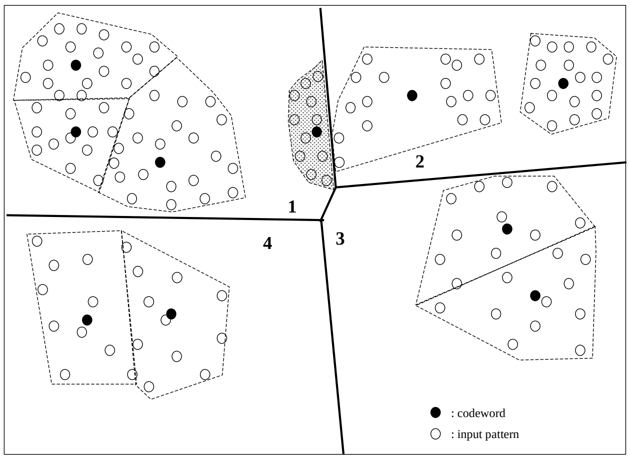

Fig. 5. The four sub-quantizers autonomously obtained by each of the four slaves

• *Migration of patterns and codewords between tasks.* The complete division of the data, on the one hand reduces the number of calculations performed and allows the tasks to work autonomously; on the other hand, prevents the patterns being compared to the whole codebook. More precisely, at each iteration of LBG and PKM, each pattern is assigned to the nearest codeword and such a research involves the whole codebook. On the contrary, in LBGS, a pattern can be represented only by the codewords that were assigned to the same slave. The result of such a local research is a final codebook that, generally, has a greater quantization error than could be obtained by the serial algorithm or by PKM, as we can see from the comparison of Fig. 5 with Fig. 6. In order to reduce the deterioration of the performances, we have introduced a new technique for the exchange of information between the processes. It allows, to some extent, the solution of these problems. Practically, it is a mechanism allowing the patterns to migrate from one process to another when it is supposed that this will produce a better result.
Let us suppose that we have the patterns shown in Fig. 4 and we divide them into four parts (as shown in the same figure). Now, let us assign each portion (macro-cell) to a different task. If no migration is allowed, the distributed

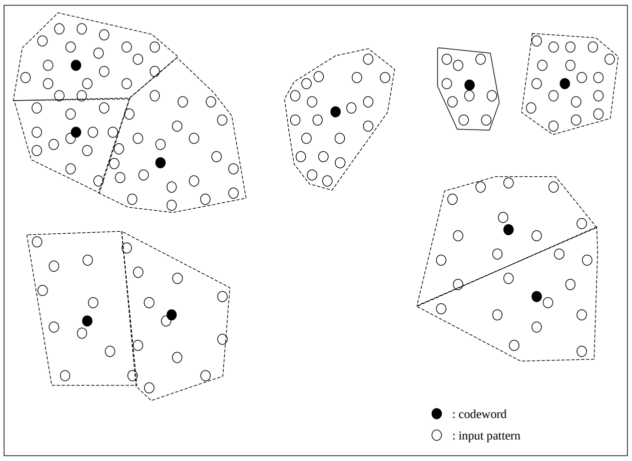

Fig. 6. Codebook obtained by LBG and PKM

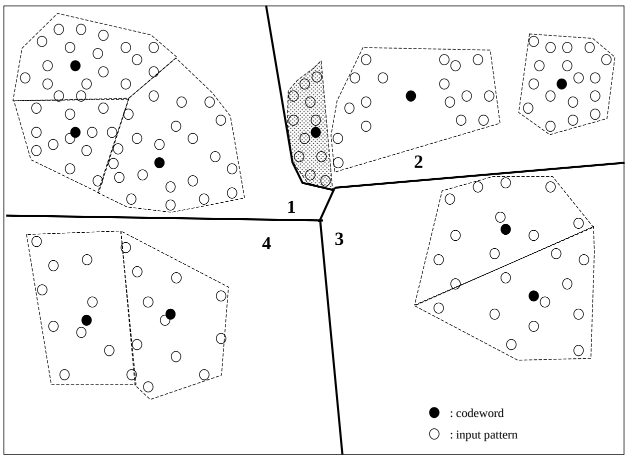

Fig. 7. Division of the patterns among the tasks after migration

algorithm, probably, reaches the configuration of Fig. 5. On the contrary, if we consider the complete technique, it is easy to reach the configuration of Fig. 7 that, in practice, has a lower global error than the configuragion in Fig. 5. Our mechanism for migration allows both patterns and codewords involved, shown in gray in Fig. 5, to "migrate" from task 1 to task 2. Fig. 7 shows the result of the migration. So, after a rearrangement, it is possible to obtain a configuration of the codewords as in Fig. 6.

- *Super-clustering and migrations of patterns and codewords.* The mechanism for identifying the cases when it is opportune that some patterns migrate from one process to another is based on the *super-clustering* of the codewords. It consists in the calculation of a *super-codebook*, i.e. a set of reference vectors representing the codebook. The elements of the new codebook are called *super-codewords.* In practice, the super-codebook is the "codebook of the codebook". It is obtained, in a distributed way, by all of the slaves and the results are shared among them. This information is used to direct the migrations of the patterns from one process to another as mentioned in the previous point. The complete mechanism, here briefly described, will be detailed in Section VII-B.

As we will see in Section VIII, at the cost of a low deterioration in performances with respect to PKM, the efficiency of LBGS is considerably increased with respect to it.

VII. Detailed description of LBGS

In this section, a more complete and detailed description of the algorithm will be given.

A. Pre-clustering and data subdivision

The first operation to be performed for the correct subdivision of data is the *pre-clustering* . After its realization, the whole IDS has been divided into N portions (or macro-cells), each represented by its own codeword (Fig. 4).

After the *pre-clustering*, the N portions of patterns (and the related codewords) determined are assigned one to each slave. Besides, each slave receives N_{C_i} codewords, constituting the portion of the codebook it has to calculate. The values for N_{C_i} have to be such that:

$$\begin{cases} \sum_{i=1}^{N} N_{C_i} = N_C \\ N_{C_i} \geq 1 \quad i = 1, \cdots, N \end{cases} \tag{8}$$

Let us observe that equation (8) can be satisfied only if $N_C \geq N$, i.e. at least one codeword for each processor is needed. We can legitimately think that the last-named condition is fulfilled because LBG has been developed for very complex problems where both the number of the input patterns and the codewords is large.

The criterion for assigning the values of N_{C_i} and its theoretical justification was suggested by Gersho's theorem [40, 41]. It says that: "*Each cell makes an equal contribution to the total distortion in optimal vector quantization with high resolution*", i.e. all cells have the same total distortion (D_i, according to eq. (4))[4].

Similarly to the concept of total distortion of a cell expressed by equation (4), we use the total distortion of a macro-cell, that is available at the end of the pre-clustering phase. According to such values, a larger number of codewords is assigned to the tasks that received macro-cells with high distortions and a smaller number of codewords to the tasks that received low-distortion macro-cells. In this (heuristic) way, we try to pursue the equalization of the total distortion of the cells as suggested by Gersho's theorem. In particular, N_{C_i} is proportional to the total distortion of the ith macro-cell.

Let us notice that the *pre-clustering* is a serial operation. However, rather than leaving its execution to a single process (for example, the master), it is more suitable that all the slaves perform it autonomously and at the same time. In the same way, each slave is able to obtain autonomously its portion of patterns and the number of codewords it has to calculate.

Once the i-th task has computed its portion of the IDS and N_{C_i}, it calculates the N_{C_i} initial codewords. More precisely, the task splits the codeword representing the macro-cell until the value N_{C_i} is reached. Now, it begins with the optimization of its codebook (of N_{C_i} elements) in an "almost" identical way to LBG. The term "almost" is used because, in order to improve the final result, the portion of the input patterns of each slave are not static, but can change as a result of patterns and codeword migrations between processess.

B. Super-clustering and codeword migrations

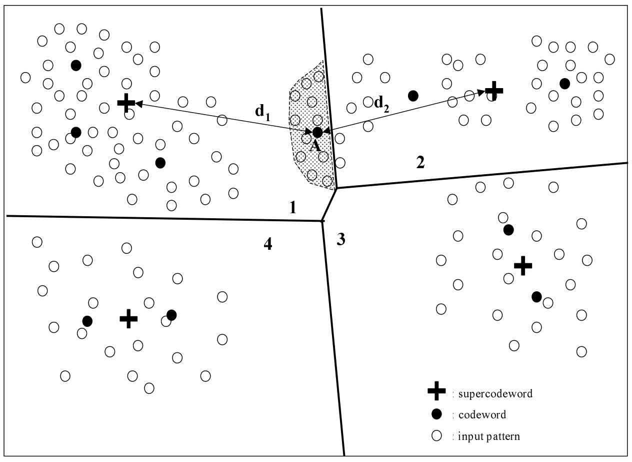

Fig. 8. Super-codewords

Let us now detail the mechanisms controlling the super-clustering of the codewords and the migration of patterns and codewords that we have already mentioned in Section VI and that allow the transition from the situation in Fig. 5 to the one in Fig. 7.

Super-clustering consists in calculating a codebook of the codebook; for this reason, we called it *super-codebook*. Practically, it is a reduced representation of the main codebook, following the same criteria shown in Section IV. Each task, at regular intervals, computes its portion of the super-codebook (i.e., it performs the super-clustering of its own portion of codewords) and communicates it also to the other tasks. In this way, all of the tasks have a global (even though approximated) vision of the distribution of all the codewords. The super-codebook is shown in Fig. 8; it is identical to Fig. 5 with the addition of the super-codewords. In this simple example, a super-codeword for each slave has been assigned. Generally, the number of the super-codewords (N_{SC}) is a configuration parameter; they are distributed among the tasks by employing the same criterion used for assigning the codewords, i.e. by equations analogous to (8)

[4]Actually, Gersho's theorem is true when certain conditions are verified (according to [40], a high-resolution quantizer has a number of codewords tending to infinite). But, in [13, 42], experimental results proved that it maintains a certain validity also for real problems where the codebook is constituted by a finite number of elements.

and assigning to each task a number of super-codewords proportional to the total distortion of the macro-cells obtained after the pre-clustering. From the tests performed, we suggest using the value $N_{SC} = N$.

At regular intervals, each task compares its codewords with the whole super-codebook. If it establishes that one of its codewords is closer to a super-codeword of another task, it transfers the whole cell in question (i.e., all of the input patterns and the codeword constituting it) to the other task and removes that cell (patterns and codeword) from its portion of data. Let us use Fig. 8 again to explain the concept better. By knowing the whole codebook, slave 1 is able to calculate that codeword A is closer to the super-codeword of slave 2 rather than to its own (in the figure, it is $d_2 < d_1$). So, it transfers the whole cell and the relative codeword (highlighted by shading) to process 2. This migration leads to the improved situation of Fig. 7.

C. Considerations regarding the temporal complexity of LBGS

Now, let us analyze the temporal complexity of LBGS and compare it with LBG and PKM. As in [20,24], we assume that, for a clustering algorithm like LBG, it is the function of the number of distance calculations between vectors. We will show that the subidvision of the data between tasks previously illustrated produces (in the ideal case) a reduction of the complexity by a factor N^2 with respect to LBG. The following scheme shows in detail the comparison in question.
- *Complexity of LBG.* In LBG, at each iteration, all of the input data set is compared with the whole codebook; so, its complexity is $O(N_P \times N_C)$.
- *Complexity of PKM.* In PKM techniques the calculation of the Voronoi partition is split among the N slaves, each comparing a portion of the input data set with the whole codebook. In the case of a perfect load balancing, its temporal complexity is, consequently, given by $O(\frac{N_P \times N_C}{N})$.
- *Complexity of LBGS.* Here, the splitting of both the input data set and the codebook among the tasks produces a further reduction of the complexity, by a factor N, with respect to PKM. However, this is true only in the case of a perfect load balancing, i.e. when the size of the portions of input patterns and codewords is the same for all of the slaves, i.e. made of $\frac{N_P}{N}$ and $\frac{N_C}{N}$ elements, respectively. In that case, the complexity would be $O(\frac{N_P \times N_C}{N^2})$.

However, we wish to underline that this value has to be simply considered as a lower-bound of the temporal complexity. But, as we will see in Section VIII, the results, in terms of speed up with respect to LBG, are very satisfactory, in particular for the most complex problems.
The analysis of the complexity performed in such terms is referred to the number of calculations per iteration. It is necessary to underline that, even by fixing N_P, N_C and N, when the configuration parameters change (see Section VIII), also the number of mean iterations executed can change with the subsequent variation in the temporal complexity of the algorithm.

D. The input parameters

From the description of the algorithm given so far, we can see that the tuning of some more parameters with respect to LBG (essentially N_C and ϵ) is required. Such parameters are related to the management of the super-clustering and the communication. Depending on the values assigned, the amount of data exchanged between the processes, the running time and the final results of the algorithm change. A list of the new input parameters is given below, together with the suggested values for each of them.
- N_{SC}: *number of super-codewords.* In conformity with the analysis of the result that will be presented in Section VIII, we suggest choosing $N_{SC} = N$;
- n_s: *number of iterations for the super-codebook computing.* This value specifies the number of iterations to be performed before a new super-clustering of the codewords occurs. In all of our tests we used the value $n_s = 3$;
- n_c: *number of iterations for the super-codebook comparison.* This value specifies the number of iterations to be performed before the slave compares its codewords again with the whole super-codebook in order to establish if any cells have to be sent to another slave; if so, they are immediately, sent. In all of our tests we used the value $n_c = 3$;

E. Evaluation of the overhead introduced by the calculation of the super-codebook

In VII-C we have shown that, in an ideal situation with a perfect load balancing, the temporal complexity of LBGS is $O(\frac{N_P \times N_C}{N^2})$. However, this formula does not consider the overhead introduced by the calculation of the super-codebook and the subsequent comparison of codebook and super-codebook.

Now, let us give an estimate of such an overhead; let us assume again that the number of distance calculations performed defines the temporal complexity of the algorithm and that the situation with a perfect load balancing is verified.
- **Super-codebook calculation**. The calculation of the super-codebook is performed by means of the optimization phase of LBG where $\frac{N_C}{N}$ codewords constitute the input patterns and $\frac{N_{SC}}{N}$ super-codewords are the reference vectors. So, an iteration of the super-clustering implies $\frac{N_C}{N} \times \frac{N_{SC}}{N}$ calculations of the distance between vectors. m_s being the number of iterations required for the convergence of the super-quantizer, if we remember that the super-codebook calculation occurs each n_s iterations of LBGS, the overhead introduced per iteration of LBGS is:

$$o_1 = \frac{N_C \times N_{SC}}{N^2} \times \frac{m_s}{n_s} \tag{9}$$

• **Comparison between codewords and super-codewords**. The comparison occurs between the $\frac{N_C}{N}$ codewords of the single slave and the N_{SC} super-codewords of the whole codebook. If we remember that this occurs each n_c iterations of LBGS, the overhead per iteration of LBGS is:

$$o_2 = \frac{N_C \times N_{SC}}{N} \times \frac{1}{n_c} \tag{10}$$

By adding o_1 and o_2, after a rearrangement of the terms, we obtain the total overhead; its value is:

$$o = o_1 + o_2 = \frac{N_C \times N_{SC}}{N^2} \left(\frac{m_s}{n_s} + N \times \frac{1}{n_c} \right) \tag{11}$$

If $a = \frac{N_P \times N_C}{N^2}$ is the original number of calculations (without super-clustering) performed by the single slave, the percentage overhead introduced by the super-clustering can be defined as:

$$\alpha = 100 \times \frac{o}{a} = 100 \times \frac{N_{SC}}{N_P} \left(\frac{m_s}{n_s} + N \times \frac{1}{n_c} \right) \tag{12}$$

For a better quantification of the overhead, we can use equation (12) and insert some typical values used for applications in the domain of LBGS (on the matter, see Section VIII):

– $N_P = 16384$;
– $N = 16$;
– $N_{SC} = 3N$;
– $m_s = 3$ (such a low value is justified by the fact that, usually, the number of super-codewords is small and so the super-clustering reaches convergence very quickly);
– $n_s = n_c = 3$;

From (12) we have $\alpha = 1.86\%$.

• **Communication complexity**

We did not evaluate the communication complexity because, in all our experiments, even when very complex tasks were executed using all the hosts, the communication overhead was negligible. Furthermore, if a more complex architecture is used we would obtain less overhead. In fact, there are no practically distributed systems or supercalculators with 20 nodes with a total maximum (and statistical) throughput of 12.5Mbytes/s with a latency in the order of microseconds as in the case of the farm of PC we used in our experimental results. As regards the processor power, in supercalculators we have not so much difference with respect to the huge difference in the network capacity.

VIII. Results

In Section III-B, we defined PISA and MISA parallel algorithms and we said that, of course, the performances of a PISA algorithm (for example, PKM and PARELBG) are identical to those of the original serial version. In that case, the quality of the implementation is only estimated by the speed-up (S), i.e. the ratio of the computing time of the serial algorithm to the computing time of the parallel one. While, for MISA algorithms (such as LBGS and PAUL), the quality has to be considered from the point of view of both performance (the final RMSE) and speed-up.

In this section, LBGS will be analyzed from this perspective. We begin with a brief discussion regarding the data sets and the configuration parameters used in the tests; after, we examine the final quantization error and the speed with respect to LBG. Lastly, several comparisons to analogous parallel algorithms are presented.

A. Some considerations common to all of the tests performed

Now, let us report some considerations that are valid for all of the tests performed. In particular, they concern the choice of the configuration parameters and the procedures for executing the tests.

To use LBGS, it is necessary to select the parameters described in Section VII-D properly. Regarding ϵ, n_s and n_c, we have employed the following values in all of our tests:

• $\epsilon = 0.001$ (see Section V-A.1);
• $n_s = n_c = 3$;

We still have to discuss the criterion for establishing the size of the supercodebook. As regards this, several elements have to be considered:

• as we saw in Section VII-E, the supercodebook should be small enough in order not to produce a large overhead;
• its size must be related also to the number of slaves because it is calculated, in a distributed way, by all of them.

Finally, after many tests, we verified that a value taking these requirements into consideration is the *mean Number of Supercordewords per Task* (N_{ST}), defined as the ratio of the total number of supercodewords (N_{SC}) to the number of the slave tasks (N). The choice of N_{ST} is made by the user in conformity to the considerations that will be presented in the next sub-sections. Here, we say briefly that, the other parameters being fixed, N_{ST} allows the user to privilege performance at the expense of speed-up and vice versa.

Regarding the distribution of the processes among the various PEs, we have used the following configuration: the master runs on the server and the slaves (one per processor) on the Intel Celeron 366-equipped machines.

Besides, all of the tests have been executed five times for each configuration and the results presented are the mean value of the five runs.

B. Analysis of the performances of LBGS

For our analysis, we chose the compression of the image we called *giraffe*[5] (984×1488 pixels). The whole picture was subdivided into square blocks of 4×4 pixels, thus obtaining an input data set of 91512 16-dimensional vectors.

Many tests were performed with this data set, trying several values for N_C, N and N_{ST}. In particular, the following values were used:

- $N_C = \{128, 256, 512, 1024, 2048, 4096\}$;
- $N = \{1, 2, 4, 8, 16\}$;
- $N_{ST} = \{0, 1, 2, 4, 8, 16\}$; $N_{ST} = 0$ is used for specifying the case with no interaction between the tasks, i.e. when, after the *pre-clustering*, each of them takes its portion of data and works in a totally independent way from the others.

Here, we report only the results related to the case $N_C = 4096$ because, with this value, the problem starts to be sufficiently complex for an objective evaluation of the performances of LBGS.

N	N_{ST}	RMSE	PSNR	$\Delta E\%$	m	T	S
1	1	7,379	42.81	0,00	10,00	5085,176	1,00
2	0	7.179	43.05	-2.72	8.00	1098.923	4.63
	1	7,162	43.07	-2,94	9,80	1255,799	4,05
	2	7,164	43.07	-2,92	9,70	1208,227	4,21
	4	7,177	43.05	-2,74	8,50	1193,015	4,26
	8	7,181	43.05	-2,69	8,80	1270,667	4,00
	16	7,181	43.05	-2,69	7,60	1091,681	4,66
4	0	7.231	42.99	-2.01	8.75	224.404	22.66
	1	7,217	43.00	-2,20	10,60	292,933	17,36
	2	7,219	43.00	-2,18	9,55	279,202	18,21
	4	7,217	43.00	-2,20	10,00	282,154	18,02
	8	7,218	43.00	-2,19	10,15	290,096	17,53
	16	7,228	42.99	-2,05	9,75	269,135	18,89
8	0	7.368	42.82	-0.15	9.75	66.233	76.78
	1	7,335	42.86	-0,60	12,05	88,288	57,60
	2	7,332	42.87	-0,65	14,58	100,696	50,50
	4	7,355	42.84	-0,34	9,70	77,690	65,45
	8	7,340	42.86	-0,53	11,65	76,568	66,41
	16	7,354	42.84	-0,34	11,03	83,430	60,95
16	0	7.305	42.90	-1.01	8.44	23.008	221.02
	1	7,217	43.00	-2,19	19,81	33,720	150,80
	2	7,228	42.99	-2,06	21,40	36,424	139,61
	4	7,255	42.96	-1,68	12,29	27,917	182,15
	8	7,271	42.94	-1,47	11,03	26,332	193,12
	16	7,281	42.93	-1,33	9,28	24,518	207,41

TABLE II
RESULTS OF LBGS FOR $N_C = 4096$ WITH *giraffe*

Fig. 9 shows the trend of RMSE with $N_C = 4096$ when the configuration changes. In the picture, we can see also a curve labeled as *ser0*. Its meaning will be explained in the next sub-sections. Equally, Fig. 10 shows the trend of the speed-up (S) for the same value of N_C ($N_C = 4096$).

Table II summarizes the results related to the compression of *giraffe* for N_C=4096.

From Fig. 10 we can see that the final results depend on both N and N_{ST}. Thus, the values reported in the columns of Table II have to be considered as functions of N and N_{ST}. In this table, several symbols appear, whose meaning has not been explained yet. They are:

[5]available at ftp://ai.unime.it/pub/data/giraffe.raw

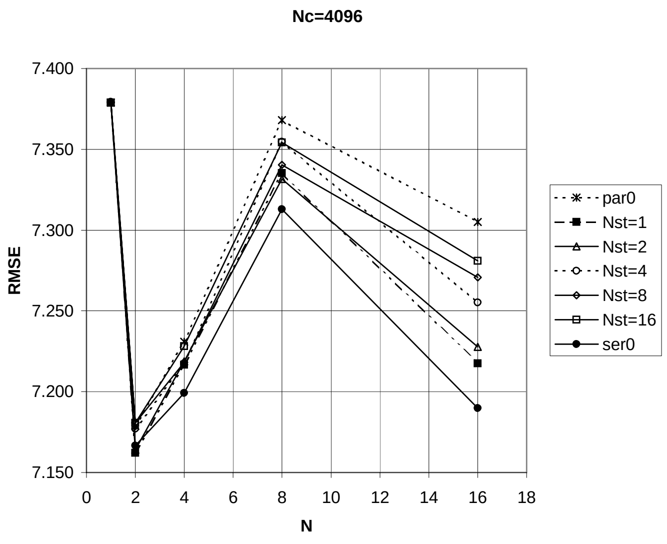

Fig. 9. RMSE; giraffe $N_C = 4096$

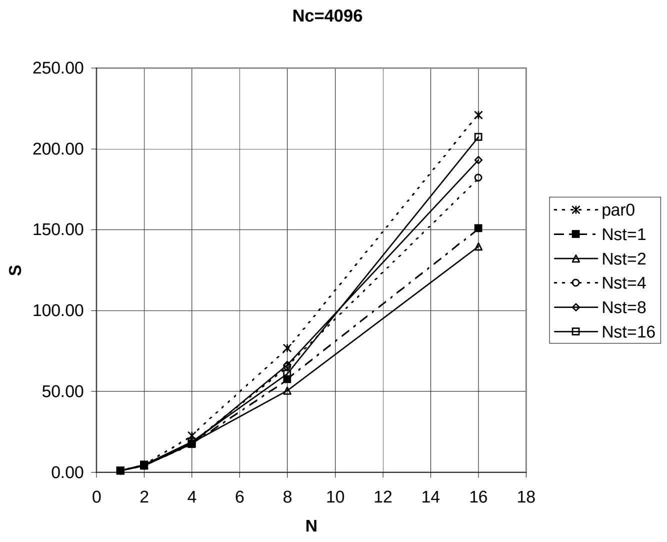

Fig. 10. Speed-up; giraffe $N_C = 4096$

• PSNR: The Peak Signal to Noise Ratio (PSNR) is often used in image applications to evaluate the resulting images after the quantization process. The PSNR is defined as follows:

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{IJ} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (f(i,j) - \hat{f}(i,j))^2} \tag{13}$$

where $f(i,j)$ and $\hat{f}(i,j)$ are respectively the grey level of the original image and the reconstructed one. All grey levels are represented with an integer value comprised in $[0, 255]$.
In our case, blocks of 4×4 pixels are considered (16-dimensional patterns) therefore $I = J = 4$ and the result of the summations in equation (13) is equal to $(RMSE)^2$. It follows that the PSNR can be evaluated as $60.17 - 20 \log_{10}(RMSE)$, that is the expression used in Table II.

• $\Delta E\%$: percentage variation of the error with respect to the serial version. Its analytical definition is:

$$\Delta E(N, N_{ST})\% = 100 \times \frac{\text{RMSE}(N, N_{ST}) - \text{RMSE}(1,1)}{\text{RMSE}(1,1)} \tag{14}$$

• m: mean number of iterations executed by the tasks;
• T: total execution time of the algorithm;
• S: speed-up of the total execution time of the algorithm with respect to the execution time of the serial version, i.e.:

$$S(N, N_{ST}) = \frac{T(1,1)}{T(N, N_{ST})} \tag{15}$$

From the analysis of the results published in the paper and of the innumerable ones not reported for the sake of saving space, we have deduced that:

- sometimes, LBGS reaches a quantization error lower than the serial version;
- even in the cases where the error of LBGS is greater than the serial version, such an increase is contained in a few percentage points;
- generally, N being fixed, when N_{ST} changes, performances and efficiency have opposite trends. In fact, larger values of RMSE match larger values of S and vice versa;
- strange but true, there are some cases where LBGS obtains speed-ups larger than N^2, i.e. the value that, in Section VII-C, had been presented as the maximum theoretical achieveable;

All of these observations will be explained and justified in detail in the next sub-sections together with an in-depth study of the dependence of the final results on the configuration parameters. This is very important because, on the basis of such knowledge, the user can decide to favour performance at the expense of speed-up or vice versa.

At first, we will study the dependence of RMSE on N and N_{ST}; afterwards, we will consider the dependence of S on the same parameters.

B.1 How RMSE changes when N changes (N_{ST} being fixed)

N_{ST} being fixed, when N increases, we expect an increase of RMSE as a consequence of splitting the problem into (almost) independent sub-problems. However, by analyzing Fig. 9 we can see a rather different trend.

To understand such behaviour, we must remember that, in an iterative optimization technique (like LBG), generally, both the initialization and the optimization phase affect the final result. We have also seen that algorithms such as ELBG, PARELBG and PAUL are virtually independent of the initialization because they include particular mechanisms to escape from the situations of local minima. On the contrary, LBGS derives from LBG and, for this reason, it is sensitive to the choice of the initial codebook.

Let us try to analyze how the two phases contribute to the achievement of the final result when N changes.

- **Optimization**. When N increases, the optimization is executed by an ever larger number of tasks working in an (almost) independent way from each other. This produces an increase of the final RMSE.
- **Initialization**. Let us remember that, in LBGS, initialization occurs in two phases (see Section VI). The former consists of the *pre-clustering*, whose result is the partitioning of the IDS into N macro-cells. Afterwards, the total distortion of each macro-cell is used to establish the number of codewords (N_{C_i}) that the ith task has to calculate. This enables us to assign a larger number of codewords to macro-cells with large distortion and a lower number where the distortion is low. Thus, it appears that when N increases, as the number of initial macro-cells increases too, a smarter initialization occurs, better than the one obtained in the serial case or for low values of N. We are going to see that such a trend is often (but not always) observed. In fact, the assignment, to each task, of a number of codewords proportional to the distortion of the original macro-cell is a very efficient heuristics but nothing ensures its optimality.

In order to estimate the effect of the initialization, the curve labeled $ser0$ has been drawn. It is obtained by initializing the codebook, for each value of N, as if it was the parallel algorithm. This is realized by executing the pre-clustering with N macro-cells, splitting each of them up to N_{C_i} and respecting (8). Afterwards, the N portions of initial codewords (each constituted by N_{C_i} elements) are joined and the optimization phase starts. This is executed serially, i.e. by a single task and not distributing the data among different tasks.

Of course, for the same values of N_C and N, the error of $ser0$ is the lowest because, with the same initialization, the serial algorithm produces the best result. The trend of $ser0$ shows exactly how the initialization affects the behaviour of the algorithm. We can see in Fig. 9 that the other curves have the same trend as $ser0$. Therefore the behaviour of the RMSE is not due to the modification introduced in the LBGS but to the original LBG and its sensitivity to different initializations.

B.2 How RMSE changes when N_{ST} changes (N being fixed)

The number of the super-codewords affects the degree of interaction between the tasks; so, this study will provide useful information about the values to employ for the parameter N_{ST}.

The first observation is that, for low values of N (N=2,4) the curves (except for $ser0$) overlap whereas they tend to split when N increases. This can be explained by considering that, for low values of N, a few patterns and codewords are in the border regions between the various group of patterns. Consequently, the interaction between the tasks (connected to the number of the super-codewords and, consequently, to N_{ST}) has a small influence on the final result. On the contrary, for larger values of N, the interaction between the tasks becomes an important factor affecting the final RMSE obtained. In that case, the following considerations are valid:

- generally, the curve related to $N_{ST} = 0$ (i.e., the case without any inter-process communication) represents the highest error; this implies that super-clustering and inter-process communication improve the quality of the final result;
- generally, the best results are obtained for low values of N_{ST} ($N_{ST} = 1, 2$), whereas RMSE increases when N_{ST} increases. This occurs because, for larger values of N_{ST} (and, consequently, of N_S), the quality of the super-clustering

improves and the super-codewords of each process are a better approximation of the related codewords with the effect of a decrease in migrations. In this way, the number of mean iterations executed by the slaves decreases and, consequently, RMSE increases. In other words, when N_{ST} increases, migrations decrease. A lower number of migrations implies that also m decreases with the consequent increase of RMSE.

B.3 Speed-up: discussion regarding the effect of N and N_{ST}

We saw in Section VII-C that, in the ideal case, the complexity of LBGS is $O(\frac{N_P \times N_C}{N^2})$. Therefore, N^2 could be considered as the maximum theoretical value achieveable for the speed-up with respect to the serial version, whose complexity is $O(N_P \times N_C)$. However, the complexity in question has been calculated by considering, in the case of a perfect load balancing, the number of distance calculations executed by each task *per iteration*. But, we know that, in LBGS, the tasks work almost independently of each other; so, even if the hypothesis of perfect load balancing is verified, each of them converges in a number of iterations that, generally, is different from the others. Besides, it is also dependent on the configuration parameters. Therefore, a meaningful evaluation of the speed-up of LBGS with respect to LBG can be performed only by considering the total execution times (see equation (15)). The employment of mean values per task and/or per iteration produces poor-utility estimates from a practical point of view.

After such considerations, we can say that N^2 has to be considered as a simply theoretical reference for the speed-up. In fact, on the one hand there are the factors discussed in Sections V-C and VII-C (serial part of the initialization, non-perfect load balancing, inter-process communication) that, by introducing overheads with respect to the ideal case, reduce the speed-up with respect to the value N^2; on the other hand, we have to consider the variation of the number of iterations executed by the processes when the configuration changes. The latter factor, can produce both an increase and a decrease of the speed-up. In this way, it is possible to justify the presence of values of S greater than N^2 in Table II.

Like RMSE, S depends on the value of N_{ST} (as well as the value of N), too. In this respect, we can make the following considerations about Fig. 10:

- N_{ST} being fixed, S increases when N increases;
- For low values of N (N=2,4) the curves (excpept for *par*0) overlap, while they tend to split when S increseases. Such behaviour, analogous to what we have seen for RMSE, can be explained in the same way: for low values of N, each macro-cell is large and the effect of the interaction between tasks is slight. On the contrary, for higher values of N, the interaction is greater and the curves tend to be different from each other.

In that case, the best results, in terms of speed-up, are obtained for high values of N_{ST}. This is because, as we have seen in Section VIII-B.2, when N_{ST} increases, the improved super-clustering leads to a reduction of the mean number of iterations executed by the slaves. This means that a reduction of the computing time occurs with the consequent increase of the speed-up. Also in this case, it is useful to analyze Table II where some useful numeric values for understanding the phenomenon are reported.

Summarizing, if N_{ST} increases, a lower number of migrations occur, m decreases and S increases.

From the analysis of Table II it is possible to observe the quite relevant results, in terms of speed-up, for the most complex problem among those considerated, i.e. the compression of *giraffe* with N_C=4096. In particular, for N=16 and N_{ST}=16, we get an execution time of 24.5 s whereas the serial execution time is 5085 s. This is a speed-up of about 207 (corresponding to a reduction in the elaboration time of 99.5%). On the contrary, if we prefer a higher quality clustering, by choosing $N_{ST} = 1$, the other configuration parameters being fixed, we get a speed-up close to 151 (reduction of 99.3% of the computing time). In both cases, for the reasons explained above (effect of a better initialization) there is a reduction in the RMSE with respect to the serial version that is quantifiable in 1.33% and 2.19%, respectively.

B.4 Conclusions

From the detailed analysis of the results we have presented, the importance of N_{ST} appears. It is a parameter the user can modify and through which decide if to privilege performance at the expense of speed-up or vice versa. Summarizing, we can say that for low values of N_{ST} (N_{ST}=1,2), better performances (lower final quantization error) and lower efficiency (lower speed-up) are obtained. When N_{ST} increases, speed-up increases, but the final quantization error increases, too.

C. Comparison with PKM, PARELBG and PAUL

Let us now compare the performance of LBGS with the performances of several previous applications implemented on the same computing system, i.e. the MULTISOFT machine. They are PKM and PARELBG, discussed before in Section V and PAUL [24], a parallel algorithm for unsupervised learning deriving from ELBG [13, 32, 33].

The comparison with PARELBG is very useful for the evaluation of the performances. In fact, as PARELEBG is the PISA-version of ELBG, the two algorithms have the the same performances that, from the numerous comparisons presented in [13], are better than or equal to the ones of the other algorithms considered.

		PARLBG			PARELBG			PAUL			LBGS		
N_C	N_T	RMSE	PSNR	S_p	RMSE	PSNR	S_p	RMSE	PSNR	S_p	RMSE	PSNR	S_p
256	1	33.744	29.61	1.00	25.815	31.93	1.00	25.815	31.93	1.00	26.571	31.68	1.00
	2	33.744	29.61	1.85	25.815	31.93	1.58	25.802	31.94	1.81	26.786	31.61	4.18
	4	33.744	29.61	3.44	25.815	31.93	2.48	25.815	31.93	3.11	26.871	31.58	15.74
	8	33.744	29.61	5.83	25.815	31.93	3.39	25.861	31.92	4.37	26.481	31.71	25.19
	16	33.744	29.61	7.51	25.815	31.93	4.00	25.908	31.90	4.59	26.689	31.64	32.62
512	1	31.751	30.14	1.00	22.508	33.12	1.00	22.508	33.12	1.00	23.649	32.69	1.00
	2	31.751	30.14	1.95	22.508	33.12	1.72	22.557	33.10	1.79	23.755	32.66	3.61
	4	31.751	30.14	3.73	22.508	33.12	2.83	22.580	33.10	3.38	23.715	32.67	13.35
	8	31.751	30.14	6.79	22.508	33.12	4.18	22.561	33.10	5.78	23.195	32.86	21.46
	16	31.751	30.14	9.02	22.508	33.12	5.32	22.604	33.09	6.04	23.256	32.84	41.94
1024	1	30.517	30.48	1.00	19.047	34.57	1.00	19.047	34.57	1.00	20.610	33.89	1.00
	2	30.517	30.48	1.98	19.047	34.57	1.84	19.037	34.58	1.74	20.680	33.86	3.57
	4	30.517	30.48	3.89	19.047	34.57	2.95	19.047	34.57	3.24	20.276	34.03	13.45
	8	30.517	30.48	7.17	19.047	34.57	4.65	19.101	34.55	5.04	19.845	34.22	27.89
	16	30.571	30.48	11.24	19.047	34.57	6.35	19.101	34.55	8.35	19.655	34.30	54.06

TABLE III
Comparison of the results obtained by PARLBG, PARELBG, PAUL and LBGS with *Lena*

		PAUL			LBGS		
N_C	N_T	RMSE	PSNR	S_p	RMSE	PSNR	S_p
128	1	13.724	37.42	1.00	13.902	37.31	1.00
	2	13.754	37.40	1.78	13.923	37.295	3.88
	4	13.737	37.41	2.43	13.984	37.257	8.77
	8	13.768	37.39	1.96	14.149	37.16	19.66
	16	13.803	37.37	1.33	14.469	36.96	19.63
256	1	11.973	38.61	1.00	12.158	38.47	1.00
	2	11.983	38.60	1.82	12.086	38.52	4.12
	4	11.988	38.60	2.87	12.173	38.46	12.06
	8	11.995	38.59	3.17	12.368	38.32	33.25
	16	12.010	38.58	2.47	12.463	38.26	31.98
512	1	10.487	39.76	1.00	10.713	39.57	1.00
	2	10.492	39.75	1.90	10.612	39.65	3.98
	4	10.498	39.75	3.47	10.661	39.61	12.72
	8	10.503	39.74	5.24	10.843	39.47	34.82
	16	10.517	39.73	4.32	10.879	39.44	50.98
1024	1	9.228	40.87	1.00	9.499	40.62	1.00
	2	9.230	40.87	1.79	9.345	40.76	3.92
	4	9.244	40.85	2.94	9.438	40.67	15.98
	8	9.244	40.85	5.24	9.555	40.57	34.91
	16	9.247	40.85	7.51	9.580	40.54	75.97
2048	1	8.078	42.02	1.00	8.402	41.68	1.00
	2	8.088	42.01	2.06	8.251	41.84	5.03
	4	8.096	42.00	4.02	8.302	41.79	19.49
	8	8.109	41.99	7.77	8.419	41.67	62.27
	16	8.115	41.98	11.51	8.379	41.71	122.04
4096	1	6.937	43.35	1.00	7.379	42.81	1.00
	2	6.943	43.34	1.90	7.162	43.07	4.05
	4	6.958	43.32	3.73	7.217	43.00	17.36
	8	6.967	43.31	7.11	7.335	42.86	57.60
	16	6.990	43.28	12.55	7.217	43.00	150.80

TABLE IV
Comparison of the results of PAUL and LBGS with *giraffe*

The first test reported in Table III compares the results obtained by PKM, PARELBG PAUL and LBGS with the famous image of *Lena* [43]. The original 8-bit gray level picture was subdivided into square blocks of 4×4 pixels, thus obtaining an input data set of 16384 16-dimensional vectors.

Actually, this task is too easy for LBGS which, as we have said many times, has been developed for very complex applications where many patterns and many codewords are involved; however, *Lena* is the most widely employed data set in the works that we deal with.

In Table III, the results otained by LBGS refer to the configuration with $N_{ST} = 1$ because it is the value that we propose to use in most cases. As illustrated in Section VIII-B, it privileges the minimization of RMSE at the expense of S.

Observations:

- in all of the cases examined, LBGS has the best speed-up;
- PARELBG and PAUL have the best performances in all of the cases considered. The performances of LBGS are a little worse but its efficiency is much higher. For example, with $N_C = 1024$ and $N = 16$, its RMSE is 2.9% higher with respect to the one obtained by PAUL (which is almost the same as PARELBG and ELBG), but the speed-up is about 6.5 times the speed-up of PAUL.

In Table IV, a comparison is reported between LBGS and PAUL with the data set *giraffe*. The same considerations made about *Lena* can be repeated.

D. Comparison with S-TREE

LBGS gets its high efficiency from the use of some techniques for reducing the number of distance calculations between patterns and codewords with respect to the number performed by LBG. In literature, other types of algorithms using similar solutions can be found. Among them, we cite the tree structured algorithms because the organization of the codebook in such techniques is similar to the hierarchical structure supercodewords-codewords of LBGS.

In particular, we consider the S-TREE algorithms [23]; they obtain better results than traditional Tree-structured Vector Quantization (TSVQ) [26], regarding both performances and efficiency.

An immediate comparison between S-TREE and LBGS is not easy to realize because LBGS was born directly as a parallel algorithm, whereas S-TREE algorithms have been presented only in serial version. For this reason, before comparing the two algorithms, it is necessary to make some considerations:

- the data set employed by the authors of S-TREE [23] for training their systems is composed of four gray-scale (8 bits) images of 256×256 pixels size; they are subdivided into square blocks of 4×4 pixels, thus obtaining, altogether, 16384 16-dimensional vectors. Such a data set is equivalent, regarding size, to our *Lena* (512×512 pixels) that we will use in this test;
- in [23], the authors report the results obtained, with the data set described at the previous point, by their S-TREE1 and S-TREE2, by the Generalized Lloyd Algorithm (GLA, the other name commonly used for LBG) and by TSVQ. Tests are repeated for several values of N_C and, for each of them, the training time and the PSNR of the codebook obtained are reported. We have used the training times to calculate the speed-up of TSVQ, S-TREE1 and S-TREE2 with respect to GLA;
- to compare the speed-up of LBGS with the ones obtained by the serial algorithms cited above, we divide the value of S obtained by LBGS (according to (15)) by the number of hosts (N) used for executing the parallel algorithm. Such a normalized speed-up can be directly compared with the values extracted from [23] where, we must remember, the algorithms are considered only in serial form.

The result of the comparison is presented in Figs. 11 and 12.

Some considerations regarding the figures:

- we have chosen the tests related to $N_C = 512$ because, for such a value, both the S-TREE algorithms and TSVQ have the highest speed-up with respect to GLA;
- the values of delta PSNR and S_{norm} of LBGS are the mean values of the results obtained with different values of N_{ST} (1,2,4,8);
- the PSNR obtained by LBGS is the best of all, that of TSVQ is the worst;
- the S_{norm} obtained by LBGS is lower than the other algorithms but, some remarks have to be made in this regard:
  - we have supposed the ideal case of linear speed-up for all of the algorithms considered (TSVQ, S-TREE1 and S-TREE2) in the event of a PISA version for them. Of course, in the real case, all of the overheads introduced by the factors described in Section V-C should be kept in mind; whereas, for LBGS, the values reported are already inclusive of such overheads;
  - in LBGS, S_{norm}, at first, increases when N increases and, after, saturates. In the ideal case (see Section VII-C), it is $S_{\text{norm}} = N$; we expect that, for more complex problems the increasing trend of S_{norm} continues to higher values of N.

In conclusion, from the comparison between S-TREE1, S-TREE2, TSVQ and LBGS we can see that, if the main requirement of the quantization task is precision, then LBGS is the algorithm to take into consideration; otherwise, the development of techniques like S-TREE on parallel systems is suggested.

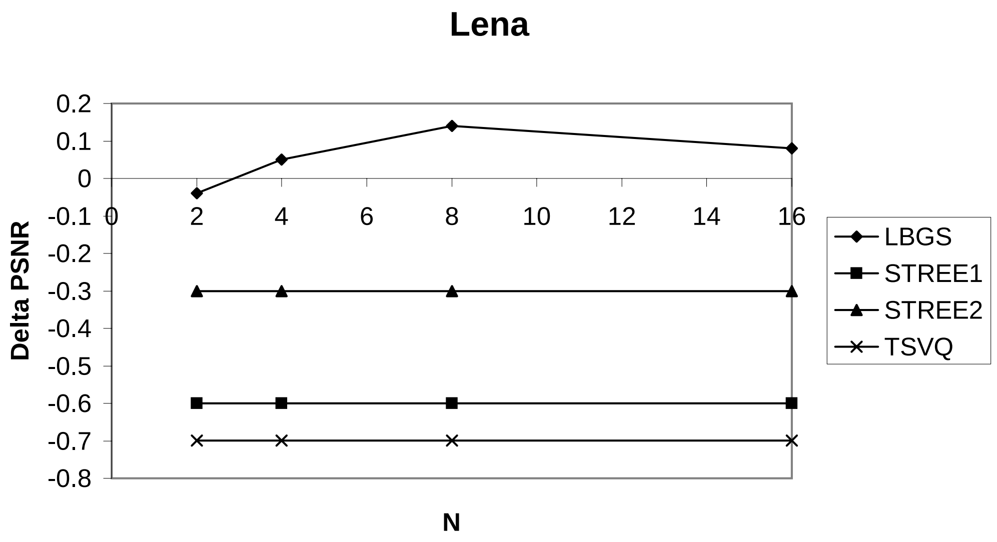

Fig. 11. Lena: delta PSNR

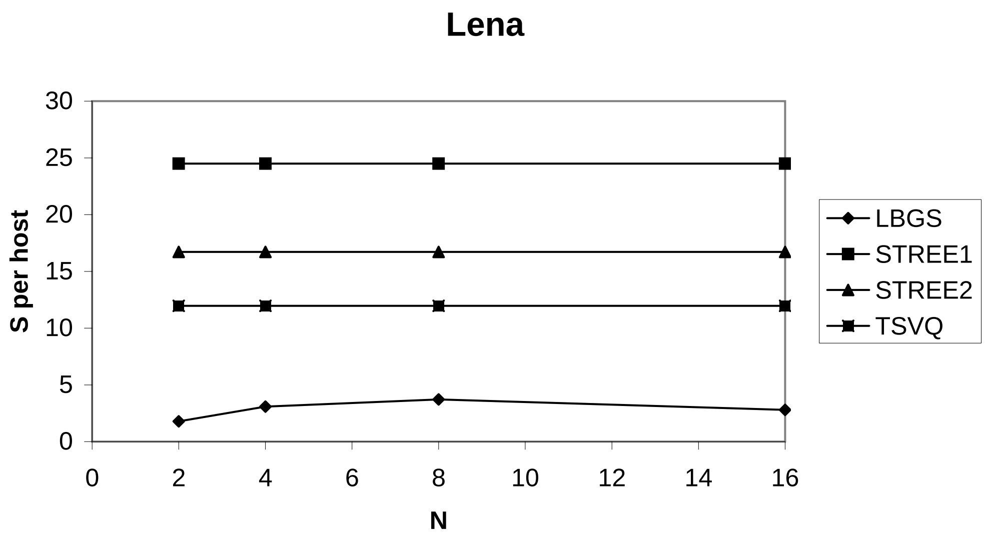

Fig. 12. Lena: speed-up

IX. Conclusions

This paper describes a new algorithm named LBGS. It derives from the well-known clustering technique called LBG and has been designed as a new proposal in the field of VQ techniques for very complex problems, where the number of both patterns and codewords is large. LBGS inserts some heuristic changes to LBG that, on the one hand allow its implementation in a very suitable way on parallel and/or distributed hardware; on the other hand, they lead to slightly worse performances, in terms of quantization error. In literature, several types of research exist in this field and LBGS is a valid instrument in those cases where it is necessary to obtain low computing times (by parallel algorithms) without an excessive deterioration of performance. Other algorithms are more inclined to parallelization (i.e., it is possible to obtain lower computing times with the same hardware and data) but, from the point of view of error, they obtain definitely a worse error. Besides, LBGS is, to some extent, configurable; so, the user can, within certain limits, privilege speed at the expense of error or vice versa. Such an interesting characteristic is rather uncommon in literature.

References

[1] A. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.

[2] H. Frigui and R. Krishnapuram, "A robust competitive clustering algorithm with applications in computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 450–465, May 1999.

[3] M. C. Clark, L. O. Hall, D. Goldgof, L. P. Clarke, R. Velthuizen, and M. S. Silbiger, "Mri segmentation using fuzzy clustering techniques," *IEEE Engineering in Medicine and Biology*, vol. 13, no. 5, pp. 730–742, 1994.

[4] J.M.Jolion, P.Meer, and S.Bataouche, “Robust Clustering with Applications in Computer Vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 791–802, Aug. 1991.

[5] S. K. Bhatia and J.S.Deogun, “Conceptual clustering in information retrieval,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 3, pp. 427–436, 1998.

[6] C. Carpineto and G. Romano, “A lattice conceptual clustering system and its application to browsing retrieval,” *Machine Learning*, vol. 24, no. 2, pp. 95–122, 1996.

[7] A.K.Jain, M.N.Murty, and P.J.Flynn, “Data clustering: A Review,” *ACM Computing Survays*, vol. 31, pp. 264–323, Sept. 1999.

[8] B.E.Watkins and M.Tummala, “Classification Vector Quantization of Image Data Using Competitive Learning,” in *IEEE International Conference on Image Processing*, vol. 3, pp. 922–925, nov 1994.

[9] N.Ling and J.R.96, “A Codebook Design Technique for better Image Quality in Vector Quantization,” in *Data Compression Conference*, p. 447, apr 1996.

[10] P.Hedelin, “Single Stage Spectral Quantization at 20 bits,” in *International Conference on Acoustics, Speech, and Signal Processing*, vol. I, pp. I525–I528, apr 1994.

[11] C.Laflamme, J.P.Adoul, R.Salami, S.Morisette, and P.Mabilleau, “16 KBPS Wideband Speech Coding Technique Based on Algebraic CELP,” in *International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 13–16, apr 1991.

[12] Y.Linde, A.Buzo, and R.M.Gray, “An Algorithm for Vector Quantizer Design,” *IEEE Transaction on Communications*, vol. 28, pp. 84–94, Jan. 1980.

[13] G. Patanè and M. Russo, “The Enhanced LBG Algorithm,” *Neural Networks*, vol. 14, pp. 1219–1237, Nov. 2001.

[14] S. Ranka and S. Sahni, “Clustering on a hypercube multicomputer,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, pp. 129–137, Apr. 1991.

[15] G. Rudolph, “Parallel clustering on a unidirectional ring,” in *Transputer Applications and Systems* (R. G. et al., ed.), vol. 1, (Amsterdam), IOS Press, 1993.

[16] F. Ancona, S. Rovetta, and R. Zunino, “Parallel architectures for vector quantization,” in *Proc. IEEE Int. Conf. on Neural Networks, ICNN’97*, vol. II, (Houston,TX), pp. 899–903, June 1997.

[17] I. Dhillon and D. Modha, “A data clustering algorithm on distributed memory machines,” in *ACM SIGKDD Workshop on Large-Scale Parallel KDD Systems*, Aug. 1999.

[18] B. Zhang, M. Hsu, and G. Forman, “Accurate recasting of parameter estimation algorithms using sufficient statistics for efficient parallel speed-up: Demonstrated for center-based data clustering algorithms,” in *4th European Conference on Principles and Practices of Knowledge Discovery in Databases (PKDD)*, Sept. 2000.

[19] G. Forman and B. Zhang, “Linear speed-up for a parallel non-approximate recasting of center-based clustering algorithms, including k-means, k-harmonic means, and em,” in *ACM SIGKDD Workshop on Distributed and Parallel Knowledge Discovery, KDD-2000*, (Boston,MA), Aug. 2000.

[20] D. Judd, P. McKinley, and A. Jain, “Large-scale parallel data clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 871–876, Aug. 1998.

[21] G.Patanè and M.Russo, “Parallel Clustering on A Commodity Supercomputer,” in *IJCNN 2000, Proc. of the IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks*, vol. 3, pp. 575–580, 2000.

[22] G. Patanè and M. Russo, “Distributed unsupervised learning using the MULTISOFT machine,” *Information Sciences*, vol. 143, pp. 1219–1237, June 2002.

[23] M. Campos and G. Carpenter, “S-TREE: self-organizing trees for data clustering and online vector quantization,” *Neural Networks*, vol. 14, pp. 505–525, May 2001.

[24] G. Campobello, G. Patanè, and M. Russo, “PAUL: a parallel algorithm for unsupervised learning,” *Signal Processing: Image Communications*, submitted.

[25] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, Jan. 1996.

[26] A.Gersho and R.M.Gray, *Vector Quantization and Signal Compression*. Boston: Kluwer, 1992.

[27] J. McQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[28] L. Ni and A. Jain, “A vlsi systolic architecture for pattern clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 80–89, Jan. 1985.

[29] D. Judd, N. Ratha, P. McKinley, J. Weng, and A. Jain, “Parallel implementation of vision algorithms on workstation clusters,” in *Proceeding. 12th International Conference on Pattern Recognition*, (Jerusalem, Israel), pp. 317–321, Oct. 1994.

[30] D. Judd, P. McKinley, and A. Jain, “Computational pruning techniques in parallel square-error clustering of large data sets,” Tech. Rep. MSU-CPS-96-02, Dept. of Computer Science, Michigan State Univ, East Lansing,Mich., 1996.

[31] A. K. Jain, R. P. W. Duin, and J. Mao, “Statistical pattern recognition: A review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[32] M.Russo and G.Patanè, “Improving the LBG Algorithm,” in *Proc. of IWANN’99* (J.Mira and J.V.Sánchez-Andrés, eds.), vol. 1606 of *Lecture Notes in Computer Science*, (Barcelona, Spain), pp. 621–630, Springer, June 1999.

[33] G.Patanè and M.Russo, “ELBG implementation,” *International Journal of Knowledge based Intelligent Engineering Systems*, vol. 4, pp. 94–109, Apr. 2000.

[34] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[35] J. Vaisey and A.Gersho, “Simulated annealing and codebook design,” in *Proceedings ICASSP’88*, pp. 1176–1179, 1988.

[36] J.C.Bezdek, “Pattern Recognition with Fuzzy Objective Function Algorithms,” in *New York: Plenum*, 1981.

[37] N.B.Karayiannis and P.-I Pai, “Fuzzy Vector Quantization Algorithms and Their Application in Image Processing,” *IEEE Transactions on Image Processing*, vol. 4, pp. 1193–1201, 1995.

[38] N.B.Karayiannis and Pin-I Pai, “Fuzzy Algorithms for Learning Vector Quantization,” *IEEE Transaction on Neural Networks*, vol. 7, pp. 1196–1211, Sept. 1996.

[39] T. Dunigan and K. Hall, “Pvm and ip multicast,” Tech. Rep. ORNL/TM-13030, Computer Science and Mathematics Divsion, Oak Ridge National Laboratory, 1996.

[40] A.Gersho, “Asymptotically Optimal Block Quantization,” *IEEE Transaction Information Theory*, vol. IT-25, no. 4, pp. 373–380, 1979.
[41] A.Gersho, *Digital Communications*, ch. Vector Quantization: A New Direction in Source Coding. North-Holland: Elsevier Science Publisher, 1986.
[42] C.Chinrungrueng and C.H. Séquin, “Optimal adaptive K-Means Algorithm with Dynamic Adjustament of Learning Rate,” *IEEE Transaction on Neural Networks*, vol. 6, pp. 157–169, Jan. 1995.
[43] D.C.Munson, Jr., “A Note on Lena,” *IEEE Transactions on Image Processing*, vol. 5, p. 3, Jan. 1996.