A Low-Cost Parallel K-Means VQ Algorithm Using Cluster Computing

Alceu de S. Britto Jr [a,b], Paulo S. L. de Souza [b], Robert Sabourin [c,d], Simone R. S. de Souza [b], Díbio L. Borges [a]

[a] Pontifícia Univ. Católica do Paraná , R. Imaculada Conceição, 1155 - Curitiba (PR) 80215-901 -Brazil
[b] Univ. Estadual de Ponta Grossa, Pr. Santos Andrade s/n, Ponta Grossa (PR) 84100-000 - Brazil
[c] École de Technologie Supérieure, 1100 Rue Notre Dame Ouest - Montreal (QC) H3C1K3 - Canada
[d] Centre for Pattern Recognition and Machine Intelligence , 1455 de Maisonneuve Blvd. West, Suite GM 606 - Montreal (QC) H3G 1M8 - Canada

Abstract

In this paper we propose a parallel approach for the K-means Vector Quantization (VQ) algorithm used in a two-stage Hidden Markov Model (HMM)-based system for recognizing handwritten numeral strings. With this parallel algorithm, based on the master/slave paradigm, we overcome two drawbacks of the sequential version: a) the time taken to create the codebook; and b) the amount of memory necessary to work with large training databases. Distributing the training samples over the slaves' local disks reduces the overhead associated with the communication process. In addition, models predicting computation and communication time have been developed. These models are useful to predict the optimal number of slaves taking into account the number of training samples and codebook size.

1 Introduction

Vector Quantization (VQ) is a widely used algorithm in image data compression, speech and writing recognition. However, the main problem related to VQ is that the training process requires large computation time and memory. For instance, the two-stage Hidden Markov Model (HMM) based method for recognizing handwritten numeral strings proposed in [1] requires the construction of 3 codebooks. When all the training samples available (19,500 digits per class from NIST SD19) are used, the time taken to create only one of these codebooks, using the K-means algorithm [2], running in an Intel Pentium III 800 MHz with 192 MB of RAM, is around 23 hours. In addition, the memory required is around 1.7 GB. This is a real problem since at each new experiment to evaluate new feature sets or increase the database for training the HMMs, it is necessary to recreate the codebooks.

Many efforts have been done to provide parallel realizations of the VQ algorithm. However, most of the time these efforts concern with the VQ encoding [3,4]. In these studies, the codevectors are distributed over the processors and the search necessary to coding an input vector is done in parallel. A parallel construction of the codebook using a distributed memory Multiple Instruction Multiple Data Stream (MIMD) architecture can be found in [5]. The authors use the master/slave method to parallelize the K-means algorithm. The training data is decomposed into patches, which are sent to the slaves by means of the network. The broadcast of these patches has a negative impact on the communication time and also it makes necessary to have enough memory on the master processor to manipulate the entire database.

This paper presents a parallel K-means VQ algorithm based on the master/slave paradigm to optimize two main bottlenecks of the sequential version: a) the time taken to create the codebook; and b) the amount of memory necessary to load the training samples. The algorithm was implemented taking into account a low-cost parallel platform based on softwares of public domain and a Cluster of Workstations (COW) composed of Personal Computers [7]. In addition, we adapted the computation model presented in [6] to predict the optimal number of slaves taking into account the number of training samples and the codebook size.

This paper is divided into 5 sections. Section 2 describes the sequential and parallel K-means algorithm. Section 3 presents the proposed computation model. Experimental results and conclusion are shown in Sections 4 and 5, respectively.

2 Vector Quantization

Let us assume that $x=[x_1, x_2, ..., x_N]$ is an N-dimensional vector whose components $\{x_k, 1 \le k \le N\}$ are real-valued. In vector quantization, the vector x is mapped onto another real-valued, N-dimensional vector y. It is used to say that x is quantized as y, and y is the quantized value of x. This can be denoted by:

$$y = q(x) \quad (1)$$

where $q(\ )$ is the quantization operator and y is the output vector corresponding to x. The value of y is a finite set of values $\{y_i, 1 \le i \le L\}$, where $y_i = [y_{i1}, y_{i2} \ldots y_{iN}]$. L is the codebook size (or number of levels), and $Y = \{y_i\}$ is the set of codevectors. To design the codebook, we partition the N-dimensional space of the random vector x into *L* regions or cells and associate with each cell C_i a vector y_i. The quantizer then assigns the code vector y_i if x is in C_i.

$$q(x) = y_i, \quad if \ x \in C_i \tag{2}$$

The mapping of x onto y results in a quantization error, and a distortion measure $d(x, y)$ can be defined between them, also known as dissimilarity. The most common distortion measure is the mean-square error, given by:

$$d(x, y) = \frac{1}{N} \sum_{k=1}^{N} (x_k - y_k)^2 \tag{3}$$

Quantization is optimal when distortion is minimized over all *L*-levels quantizers. There are two necessary conditions for optimality. The first condition is that the optimal quantizer is realized by using a minimum-distortion or nearest neighbor selection rule.

$$q(x) = y_i, \quad iff \ d(x, y_i) \le d(x, y_j), \ j \ne i, \ 1 \le j \le L \tag{4}$$

This means that the quantizer selects the code vector that results in the minimum distortion with respect to x. The second necessary condition for optimality is that each code vector y_i is chosen to minimize the average distortion in cell C_i. Let us consider $\{x(n), 1 \le n \le M\}$ as a set of training vectors, and M_i as a subset of those vectors in cell C_i. The average distortion D_i is then given by:

$$D_i = \frac{1}{M_i} \sum_{x \in C_i} d(x, y_i) \tag{5}$$

The vector that minimizes the average distortion in cell C_i is called the centroid of C_i, and it is denoted as:

$$y_i = cent(C_i) \tag{6}$$

2.1 Standard K-Means Algorithm

One well-known method for codebook design is an iterative clustering algorithm known in the pattern recognition literature as the K-means algorithm [2], where $K = L$ (codebook size). The algorithm divides the set of training vectors $\{x(n)\}$ into *L* clusters C_i in such a way that the two necessary conditions for optimality are satisfied. In the algorithm description, m is the iteration index and $C_i(m)$ is the i[th] cluster at iteration m, with $y_i(m)$ its centroid. The algorithm is shown in Figure 1.

1. **Initialization Step:** Load the training set $\{x(n), 1 \le n \le M\}$ into memory. Set $m = 0$. Choose a set of initial code vectors $y_i(0), 1 \le i \le L$.
2. Classification Step: Classify the set of training vectors $\{x(n), 1 \le n \le M\}$ into the clusters C_i by the nearest neighbor rule. $x \in c_i(m), \ iff \ d(x, y_i) \le d(x, y_j), \ all \ j \ne i, \ 1 \le j \le L$
3. Code Vector Updating Step: $m = m + 1$. Update the code vector of each cluster by computing the centroid of the corresponding training vectors in each cluster $y_i(m) = cen\{C_i(m)\}, \ 1 \le i \le L$.
4. Termination Test Step: If the decrease in the overall distortion $D(m)$ at iteration m relative to $D(m-1)$ is below a certain threshold, stop; otherwise go to the Classification Step.

Figure 1 – Sequential K-means Algorithm

2.2 Parallel K-Means Algorithm

The parallel version of the K-Means algorithm is shown in Figure 2. The set of training samples $x(n)$ was divided into *S* portions of size *M/S*, where *S* is the number of slaves and *M* is the number of individuals at the entire set of training samples. In this way, each slave works with its subset of training samples and calculate three partial vectors: $f_i^w(m+1)$, $z_i^w(m+1)$ and $t_i^w(m+1)$, respectively, partial distortion, partial sum and partial incidence (see step 2 – Figure 2). The master process uses these three partial vectors in order to generate a global version of them. With the global values, the master calculates the new code vector and determines the new distortion (see steps 3 and 4 – Figure 2).

The subset of samples loaded by each slave is brought from the local hard disk in order to minimize the cost caused by the communication network and also to reduce the quantity of memory necessary at the master machine. This project option brings about three important advantages. First, it minimizes the network access, because the data were gathered directly from the local hard disk. Second, it takes off the needs of loading all training samples in a unique node (now this file is partially loaded in each slave). Finally, it optimizes the loading of the training samples, since the slaves do this activity now in a parallel way.

1. Initialization Step:

The master sets $m = 0$ and chooses a set of initial code vectors $y_i(0), 1 \le i \le L$.

2. Classification Step:

$\Rightarrow$ Each slave $w, 1 \le w \le S$, just at the first iteration, load its subset of training vectors $\left\{ x_w(n), 1 \le n \le \frac{M}{S} \right\}$.

$\Rightarrow$ Each slave w, classifies a subset of training vectors into the cluster C_i by the nearest neighbor rule.

$$x_w \in c_i(m), \quad iff\, d(x_w, y_i) \le d(x_w, y_j), \quad all\, j \ne i, \; 1 \le j \le L$$

$\Rightarrow$ Each slave w builds a partial centroid updating, considering $x_w \in c_i(m)$

$$f_i^w(m+1) = f_i^w(m+1) + d(x_w, y_j);$$

$$z_i^w(m+1) = z_i^w(m+1) + x_w;$$

$$t_i^w(m+1) = t_i^w(m+1) + 1;$$

3. Code Vector Updating Step:

The master updates the code vector of each cluster by summing the partial values from slaves and computing the centroid of the corresponding training vectors in each cluster.

$m = m + 1$,

$$Z_i = \sum_{w=1}^{S} z_i^w(m), \quad T_i = \sum_{w=1}^{S} t_i^w(m), \quad F_i = \sum_{w=1}^{S} f_i^w(m)$$

$$y_i(m) = \frac{Z_i}{T_i}, 1 \le i \le L$$

4. Termination Test Step:

The master calculates the overall distortion D(m), according to:

$$D(m) = \frac{1}{M} \sum_{j=1}^{L} F_j(m)$$

If the decrease in the overall distortion $D(m)$ at iteration m relative to $D(m-1)$ is below a certain threshold, the master stops and kill all the slaves; otherwise it goes back to the Classification Step.

Figure 2 – Proposed Parallel K-Means Algorithm

These three advantages make a fourth one: the scalability is favored. This happen because the communication is reduced becoming dependent mainly on the size of the code vectors and not on the number of samples (this last obviously a higher value). The disadvantage of this option is the necessity of having a hard disk in each node with enough free space to store *M*/*S* samples. These features can be better observed by means of pseudo-code showed in the Figure 3.

The natural sequence of instructions of the master algorithm should be: send the current codebook to the slaves; receive the new partial values from the slaves; calculate the new codebook and calculate the distortion between the early codebook and the new one. However, this sequential order do not allow to employ the correct parallelism, because while the distortion calculation is being done, the slave remain in stand by, waiting for the new codebook. Taking into account that just the last iteration will cause an unnecessary start of the slaves, it was chosen to change the order of master algorithm (see Figure 3). Thus, the master code responsible for calculating the new codebook from partial values, when executing precedes the code responsible for distortion calculation. In the master algorithm can be seen that as soon as the new codebook is known it is sent again to the slaves. While the slaves start the calculation of next iteration, the master process verifies the distortion. When the distortion is lower than a threshold, the slave processes are killed by the master, the current codebook at master process is saved and the application is ended. The calculation done by slaves at the last iteration is lost.

```
Master Algorithm
choose a set of initial code vectors
while (true)
   for each slave w,  1 ≤ w ≤ S
     send the set of centroids  to w^th slave
   end for
   if not first iteration
     calculate new distortion
     if (old_distortion – new_distortion) <= threshold
       kill slaves
       save centroids
       exit application
     end if
   end if
   for each slave w,  1 ≤ w ≤ S
     receive from w^th slave partial values of:
        distortion, sum and incidence
     sum partial values
   end for
   calculate new centroids with global values
end while
```

```
w^th Slave Algorithm, for 1 ≤ w ≤ S
load w^th slice of samples from local hard disk
while(true)
   receive centroids from master
   calculate partial centroids for w^th slice of samples:
      generate partial: distortion, sum and incidence
   send to master partial values of:
      distortion, sum and incidence
end while
```

Figure 3 – The algorithm used by master and slave processes.

3 Computation model

Let us consider T_{iter} as the time needed to calculate the codebook change after presentation of all samples. This represents the time consumed to calculate the distortions in order to classify the training samples at each iteration of the 2nd Step (Classification) of the Figure 1. *M* as the number of training samples; *L* as the codebook size; and T_{update} as the time needed to execute the 3rd Step (Code Vector Updating) of the Figure 1. Thus, the

computation model for the sequential K-means algorithm can be written as:

$$T_{sequential} = (\alpha_{sample} \times M + \alpha_{update}) \times L \quad (7)$$

where,

$$\alpha_{sample} = \frac{T_{iter}}{M \times L}, \quad \text{and} \quad \alpha_{update} = \frac{T_{update}}{L} \quad (8)$$

where, α_{sample} is the time required to compute one training sample at the 2nd Step of the Sequential K-means algorithm; and α_{update} is the time needed to update one codevector of the codebook.

To predict the performance of the proposed parallel version it is necessary to consider both the computation and communication times. The communication model depends on the number of slaves and the message size. This can be written as:

$$T_c = F + V \times size \quad (9)$$

where, F is a fixed overhead cost associated with any communication, V is a variable cost, and *size* is the size of the codebook in bytes. Thus, we can model the parallel total time as:

$$T_{parallel}(S) = \left(\alpha_{sample} \times \frac{M}{S} + \alpha_{update} \right) \times L + Tc \times S \quad (10)$$

where, S is the number of slaves. As we can see as more slaves are used the computation time decreases, but at the same time, the communication time increases. However, since the data distribution used avoids sending the data through the network the impact of communication time is reduced. Basically, the *Tc* corresponds just to the time needed to send the codebook from slaves to master and *vice-versa.*

The ratio of the execution time when using one node to the one when S nodes are used is defined as Speedup [7], and can be written as:

$$Speedup(S) = \frac{T_{sequential}}{T_{parallel(S)}} \quad (11)$$

The *Speedup(S)* can be normalized taking into account the number of nodes. This metric is known as Efficiency[7], and can be written as:

$$E(S) = \frac{Speedup(S)}{S} \quad (12)$$

Finally, the optimal number of slaves S^* can be determined by considering two metrics. The first one is based on the maximum efficiency *E(S)*, and can be written as:

$$S^* = \frac{\sqrt{V \times size + \alpha_{sample} \times M}}{F} \quad (13)$$

The second one is based on the total execution time, which can be minimized when the efficiency *E(S)* = 50%. In this case, even using just 50% of the platform power it is possible to reach the lowest total execution time.

4 Experimental results

The proposed algorithm was executed on a heterogeneous low-cost cluster of workstations (COW) composed of 10 PCs, each one with its own address space (distributed memory). The operating system used was the Linux 2.4.5 and an IEEE 802.3 10BASET network interconnected the nodes. Table 1 shows a description of the used nodes. This platform has a maximal theoretical peak performance of 8.3 Gflops. The parallel application was developed using the C language and the message-passing environment Parallel Virtual Machine (PVM).

Table 1 - COW configuration. C = category; T = total number of machines on the category.

C	T	Description	RAM	GFLOPS
1	2	Intel Pentium III 800 MHz	192MB	0.8
2	1	AMD Duron 950 MHz	128MB	0.9
3	7	Intel Pentium III 733 MHz	128MB	0.7

To evaluate the proposed parallel K-means algorithm and also our computation model we can consider the construction of one of the codebooks necessary to the method for recognizing handwritten numeral strings proposed in [1]. In this experiment, $M = 1{,}709{,}156$; $L = 256$; α_{sample} and α_{update} were empirically defined as 1.4×10^{-6} s and 2.58203×10^{-6} s, respectively. Using the analysis presented in the previous section, it can predict the computational time of the sequential K-means algorithm as:

$$T_{sequential} = (1.4 \times 10^{-6} \times 1{,}709{,}156 + 2.58203 \times 10^{-6}) \times 256$$

$$T_{sequential} = 612.562\, s$$

To determine the communication time, the values of F and V were also experimentally defined as $0.02s$ and $1.06591 \times 10^{-6} s$, respectively. The message size (48,128 bytes) is computed as the number of entries in the codebook (L=256) multiplied by the number of entries in each codevector (N=47) and multiplied by the size in bytes of the data type (float = 4). Thus, the communication time can be estimated as:

$$T_c = 0.02 + 1.06591 \times 10^{-6} \times 48{,}128 = 0.0713\, s$$

Thus, the computation time of the parallel K-means algorithm considering 10 slaves can be predicted as:

$$T_{parallel} = \left(1.35\times10^{-6}\times\frac{1{,}709{,}156}{10} + 2.58203\times10^{-6}\right)\times 256 + 0.0713\times10 = 59.782\,s$$

Here it is possible to observe that α_{sample} is 1.35×10^{-6} instead of 1.4×10^{-6} used to calculate the sequential execution time. This is due to the lower swap activity observed on the slaves when compared to that observed when just one processor was used to manipulate the entire training data. The predicted speedup is this case is 10.25.

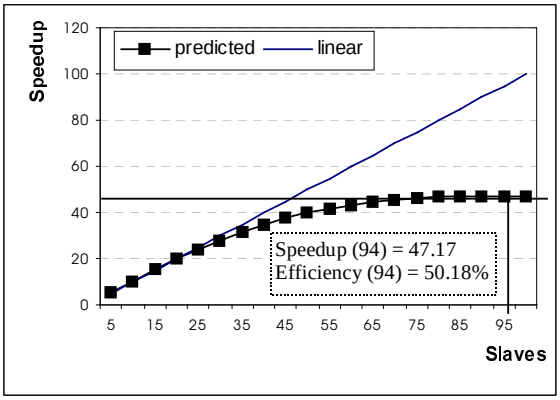

Figure 4: Predicted speedup for different number of slaves

The optimal number of slaves can be predicted as:

$$S^* = \sqrt{\frac{1.06591\times10^{-6}\times48{,}128 + 1.35\times10^{-6}\times1{,}709{,}156}{0.02}} = 10.86$$

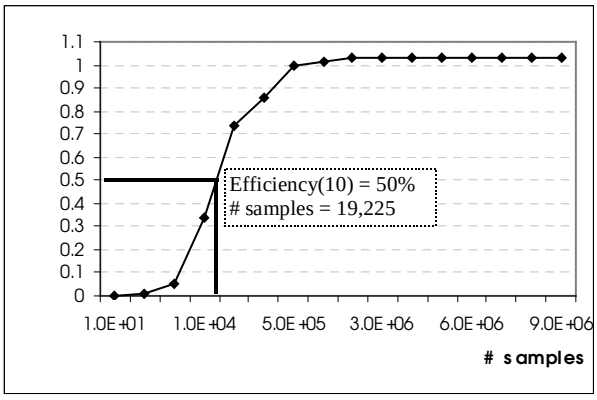

Figure 5: Efficiency taking into accounts 10 slaves and different number of samples

In a real experiment using the COW described in this section (10 slaves), a speedup of 10.11 was observed. Figure 4 shows the speedup predicted based on the proposed models considering 1,709,156 training samples, 256 codevectors and a different number of slaves. As we can see, the optimal number of slaves that minimize the total execution time is 94. However, if the objective is to maximize the efficiency, the number of slaves estimated by equation (13) is around 11.

Figure 5 shows the predicted efficiency taking into account the use of 10 slaves and a different number of training samples (from 100 to 9,000,000). It is possible to observe that for a number of samples lower than 19,225 the number of slaves must be decreased according to equation (13).

In a last experiment, the COW previously described (10 slaves) was used to calculate the codebook necessary to train our string recognition system using the entire NIST SD19 (M=8,844,798 and L=256). The total execution time observed was around 2h:20min.

5 Conclusion and future works

A low-cost parallel approach for the K-means VQ algorithm based on the master/slave paradigm and cluster computing was proposed. It allowed to overcome two drawbacks of the sequential version: a) the time taken to create the codebook; and b) the amount of memory necessary to work with large training databases. Distributing the training samples over the slaves' local disks has shown to be an interesting strategy to reduce the overhead associated with the communication process. The computation and communication time were predicted within a very small margin of error.

References

[1] Britto A.S., Sabourin R., Bortolozzi F. and Suen C.Y., A two-stage HMM-based system for recognizing handwritten numeral strings. Proc. of the Inter. Conf. on Document Analysis and Recognition, Seattle, USA, 2001, pp. 396-400.

[2] MacQueen J. Some methods for classifications and analysis of multivariate observations. Proc. of Fifth Berkeley Symposium on Math, Stat. and Prob., 1967, pp. 281-297.

[3] Manohar M., Tilton J. Progressive vector quantization of multispectral image data using massively parallel SIMD machine. Data Compressions Conference, Snowbird, UT, 1992, pp. 181-186.

[4] Kobayashi K. et al. A memory based parallel processor for vector quantization: FMPP-VQ, IEICE Trans. Electron. E80-C7, 1997, pp. 970-975.

[5] Abbas H.M., Bayoumi, M.M. Parallel codebook design for vector quantization on a message passing MIMD architecture. Parallel Computing, 28, 2002, pp. 1079-1093.

[6] Cantú-Paz, E. Designing Efficient Master-Slave Parallel Genetic Algorithms. IlliGal Report No. 97004, Illinois Genetic Algorithms Laboratory, 1997.

[7] Hwang K., Xu Z. Scalable parallel computing. McGraw-Hill: New York, USA, 1997, 802p.