Parallelization Of The LBG Vector Quantization Algorithm For Shared Memory Systems

Rajashekar Annaji annaji.rajashekar@iiitb.ac.in
International Institute of Information Technology
Bangalore, 560 100
India

Shrisha Rao srao@iiitb.ac.in
International Institute of Information Technology
Bangalore, 560 100
India

Abstract

This paper proposes a parallel approach for the Vector Quantization (VQ) problem in image processing. VQ deals with codebook generation from the input training data set and replacement of any arbitrary data with the nearest codevector. Most of the efforts in VQ have been directed towards designing parallel search algorithms for the codebook, and little has hitherto been done in evolving a parallelized procedure to obtain an optimum codebook. This parallel algorithm addresses the problem of designing an optimum codebook using the traditional LBG type of vector quantization algorithm for shared memory systems and for the efficient usage of parallel processors. Using the codebook formed from a training set, any arbitrary input data is replaced with the nearest codevector from the codebook. The effectiveness of the proposed algorithm is indicated.

Keywords: Vector Quantization, Multi-core, Shared Memory, Clustering.

1 INTRODUCTION

Vector Quantization is an extension of the scalar quantization to multi-dimensional space [7], which is a widely used compression technique for speech and image coding systems [4]. It is similar to the clustering procedure known as K-means algorithm [10] which is used in pattern recognition. It is also useful in estimating the probability distributions of the given featured vectors over a higher dimension space.

For Data compression using Vector Quantization, the two main distinguishable parts are codebook generation and replacement of arbitrary data with the obtained codebook. This paper addresses the parallel implementation of the codebook generation part.

Vector quantization involves the comparison of vectors in the input training sequence with all the vectors in a codebook. Owing to the fact that all the source vectors are compared with the same codebook and the source (training) vectors are mutually exclusive in operation, division of work can be clearly visualized and parallelization of the algorithm can be achieved.

The LBG algorithm used for Vector Quantization needs to be modified by exploiting the inherent parallelism, which should lead to the improvement in the usage of processors. And this will in turn reduce the time taken to generate the codebook for LBG algorithm.

This paper presents a parallel VQ algorithm based on the shared memory architecture and uses the master/slave paradigm to optimize two main bottlenecks of the sequential version:

- Time taken for the design of optimum codebook.
- Efficient distribution of work to the parallel processing units.

Taking the advantage of shared memory architecture by assigning equal chunks of input training data to all the processors and having a copy of codebook in the primary memory of each processor, the 'nearest codevector identification' which is the most time consuming part in the LBG algorithm is performed faster. The reduction in computation time does not result in the increase in distortion. Distributing the training samples over the local disks of slaves (processors) reduces the overhead associated with the communication process.

The 'cell table' which is formed after the integration by the master is stored in shared memory and is used in the Updating procedure. This is important for obtaining optimum codebook for the given input training data. Processors work parallel, updating a single codevector at any point. Assigning of the codevector to a processor is done randomly.

The paper is organized as follows. Section 2 describes the related work of vector quantization in the field of data compression and also provides some background about LBG and terms used in the algorithm. Section 3 describes the proposed Parallel implementation of Vector Quantization algorithm. Section 4 describes the performance issues of the above proposed algorithm and speedup of the system. Section 5 describes the results and simulations of the algorithm using OpenMP. Section 6 concludes the work done and describes some of the future work.

Related Work

Considerable work is being done in the fields of image compression, speech compression based on vector quantization and codebook generation. Some of the algorithms that have been used for sequential VQ codebook generation are LBG, pair wise nearest neighbor (PNN), simulated annealing, and fuzzy c-means clustering [10] analysis. The compression method2 used for this is based on a block encoding scheme known as vector quantization [1, 9, 10]. The last decade has seen much activity in exploring limits and applications for vector quantization. Currently, parallel processing is being explored as a way to provide computation speeds necessary for real time applications of image compression techniques [4].

Codebook design plays a fundamental role in the performance of signal compression systems based on VQ. The amount of compression is defined in terms of the rate, which will be measured in bits per sample. Suppose we have a codebook of size N , and the input vector is of dimension L, in order to inform the decoder of which code-vector was selected, we need to use $log2N$ bits. Thus the number of bits per vector is $log2N$. As each codevector contains the reconstruction values for L source output samples, the number of bits per sample would be $\frac{Log_2 N}{L}$. Thus, the rate for an L dimensional VQ with a codebook size of N is $\frac{Log_2 N}{L}$.

The main problem related to VQ is that the training process to create the codebook requires large computation time and memory, since at each new experiment to evaluate new feature sets or increase in the database for training the HMM's(Hidden Markov Models), it is necessary to recreate the codebooks. Parallelism operates on the fact that large problems can almost always be divided into smaller ones, which may be carried out concurrently. Based on this principle, an algorithm for parallelizing of Vector Quantization (VQ) is proposed, which when applied on a Shared Memory system like a Multi-core system guarantees a better and faster initialization method for LBG codebook design.

Codebook consists of a set of vectors, called codevectors. Vector quantization algorithms use this codebook to map an input vector to the codevector closest to it. Data compression, the goal of vector quantization, can then be achieved by transmitting or storing only the index of the vector. Various algorithms have been developed to generate codebooks. The most commonly Known and used algorithm is the Linde-Buzo-Gray (LBG) algorithm.

Parallel versions of the LBG Vector Quantization algorithm have been proposed by many and most of them have been applied to Distributed systems where the overhead of process communication is present [4, 7, 6]. The speed and execution time of the algorithm depends on these communications mechanisms, which are minimal in case of shared memory architectures. Though some parts of these parallel implementations are similar, the speed and effectiveness depends upon the architecture used and efficient usage of system processors and memories.

Algorithms for Codebook Generation

Various algorithms have been developed for codebook generation. Some that have been used are the LBG, pair wise nearest neighbor (PNN), simulated annealing and the fuzzy c-means (FCM) clustering algorithms.

1. LBG Algorithm [1, 9]: This is an iterative clustering descent algorithm. Its basic function is to minimize the distance between the training vector and the code vector that is closest to it.

2. Pair wise Nearest Neighbor Algorithm [3]: This is a new alternative to the LBG algorithm and can be considered as an initialize for the LBG algorithm. For efficient execution of this algorithm, the two closest training vectors have to be found and clustered. The two closest clusters are then merged to form one cluster and so on.

3. Simulated Annealing Algorithm: This algorithm aims to find a globally optimum codebook as compared to the locally optimum codebook that is obtained using the LBG algorithm. Stochastic hill climbing is used to find a solution closer to the global minimum and escape the poor local minima.

4. Fuzzy C-Means Clustering Algorithm [10, 9]: The LBG and PNN partition the training vectors into disjoint sets. The FCM algorithm assigns each training vector a membership function which indicates the degree to which it will belong to each cluster rather than assigning it to one cluster as is done in the LBG and PNN algorithms.

Huang compares these algorithms [9]; the following are some of his conclusions.

- PNN is the fastest but has higher distortion than the LBG algorithm [3].

- Simulated Annealing produces the best distortion results but requires substantially greater CPU time and there is no significant improvement in the quality of the reproduced images.

- The FCM algorithm has worse results and is slower than the LBG Algorithm.

Background

Looking at the different codebook generations and their drawbacks, the algorithm chosen for parallelizing was the LBG algorithm. An attempt has been made to decrease the time for codebook generation while maintaining the distortion measures obtained from the sequential version of the LBG algorithm. The next section describes the LBG algorithm in more detail which is taken from the original Vector Quantization paper [1].

LBG Algorithm

- Initialization: Given N = number of levels, distortion threshold € ≥ 0 , an initial N - level reproduction alphabet $A0$ and a training sequence { xj ; j = 0... n – 1}. Set m = 0 and $D-1 = \infty$.

- Given $A_m = \{yi ; i = 1....N\}$ find the minimum distortion partition $P(A_m) = \{Si; i = 1...N\}$ of the training sequence: $xj \in Si$ if $d(xj, yi) \leq d(xj, yl)$, for all l . Compute the average distortion

$$Dm = D(\{A_m, P(A_m)\}) = n^{-1}\sum_{j=0}^{n-1} min_{y \in A_m} d(x_j, y).$$

- If $\frac{D_{m-1} - D_m}{D_m}$ halt with Am final reproduction alphabet. Otherwise continue.

- Find the optimal reproduction alphabet $x(P(Am)) = \{x(Si); i = 1, \ldots, N\}$ for $P(Am)$. Set $A_{m+1} \equiv x(P(Am))$. Replace m by $m+1$ and go to 1 .

Terms used:

- Codebook: It is a collection of codevectors, and these codevectors can be stated as the quantization levels. In LBG algorithm number of codevectors can be only in powers of 2 .

- Centroid: Centroid is nothing but the average of the input vectors in the particular region specified. The dimension of the centroid is same as Input training vector k

- Partial Cell Table: It is the part of the 'cell table' and it indicates the allocation of input vectors to the corresponding minimum distortion codebook. Against each index it stores the codevector number from which the corresponding input vector has the minimum distance (Euclidean distance). Each processor has its own partial cell table.

- Cell Table: After all the processors compute the minimum distortions and partial cell tables are formed, they are integrated to form the final 'cell table' and the values in the table are called as cell value.

- Distortion: The Euclidean distance between the input training vector and the codebook vector gives the distortion value. It helps indentifying the nearest codebook.

2 System Model for Parallel implementation of Vector Quantization

This parallel algorithm can be extended to shared memory architectures where in, all the processor cores have their primary cache memory, also called as $L1$ cache. And a single shared memory also called as $L2$ cache, which is accessible by all the cores by some means inter-process communication. The input training data is available in the shared memory and hence can be shared by all the processors.

Notations

We assume that the initial training data which is used to train the system is of the form of a $'M \times k'$ matrix where $'k'$ is the vector dimension. This can be any numerical data, which is obtained by processing an image or a speech signal.

- Number of Input Training vectors : M
- The dimension of Input data: $1 \times k$
- Codebook size: $N \times k$, where N is the number of codevectors
- Number of processors: P

The set of training vectors are
$\tau = \{X1, \ldots, XM\}$

And each input vector is denoted by,

$Xm = \{xm1, \ldots, xmk\}, m = 1, 2, \ldots, M$

$C = \{C1, \ldots, CN\}$

represents the codebook. Each codevector is k -dimensional, e.g.

$Cn = \{cn1, \ldots, cnk\}, n = 1, 2, \ldots, N$

Algorithm Implementation

The parallel algorithm is given as follows.

Functions used:

Centroid (Input) → The average of input.

CodebookIndex (MinDist) → this function gives the index of the codevector to which the input is nearest and the argument is “Minimum Euclidean distance”.

CommunicateToMaster () → All the processors communicate to the master to integrate the parallel sections.

IntegratePartialCellTables () → the partial cell tables formed by individual processors are integrated to form the final ’cell table’.

IntegrateDistortions () → the distortion value obtained by each processor for its set of input vectors allocated is communicated to the master, which does the integration of all those distortions.

ExtractVectors(CellTable, index) → the values in the ‘CellTable’ are the indexes of codevectors. From the ’CellTable’ extract all the input vectors which belong to a particular codevector denoted by its index.

NewCodevector (extracted vectors) → the centroid of the extracted vectors gives the updated codevector.

Input: Any data of dimension $M \times k$

Output: Codebook of dimension $N \times k$

```
MasterSelection ();
            → compute centroid (input);
            → Distribute inputs to Processors
 Codebook Splitting:
            foreach $centroid$ do $[centroid + \delta], [centroid - \delta]$

Cell Allocation:
            → parallel execution at each processor with processor ID $\rho = 0, 1, \ldots, P - 1;$
            {
            $for\ z \leftarrow [\ \rho \times (M)/P + 1]\ to\ [(\rho + 1) \times M/(P)$ ] do
                  MinDist ←
                $minimum(E.Dist[input(z), codebook(0)], E.Dist[input(z), codebook(1)] \ldots);$
                $index \leftarrow CodebookIndex(MinDist);$
                $D_p \leftarrow D_p + MinDist;$
                PartialCellTable (z) → index ;
             end
            }

CommunicateToMaster ();
 celltable () → IntegratePartialCellTables ();
$TD_p \leftarrow$ Integrate Distortions ();
if $\frac{TD_p - TD_{p-1}}{TD_p} \leq €$ then
            if $Vectors == N$ (required number of codevectors) then
                        TERMINATE;
             else go to CodebookSplitting step
end
else Updation step:
→ parallel execution at each processor;
{
for $j \leftarrow 1\ to\ M$ do
            Extract Vectors ($CellTable, index$);
            NewCodevector ($index$) ← centroid ($Extracted\ Vectors$);
end
 }
```

```
    go to Cell Allocation step;
```

Algorithm 1: Parallelized Vector Quantization Algorithm

The following are the important steps in the Algorithm:

1. Initialization
2. Codebook Splitting
3. Cell Allocation
4. Updation

Initialization:

- One of the processors is chosen as the master either by leader election or randomly.
- The master computes the centroid of the initial training data and it is assigned as the initial codebook.

Input Training Set					
1	a_{11}	a_{12}	……	a_{1k}
2	a_{21}	a_{22}	……	a_{2k}
:				
:	a_{m1}	a_{m2}	……	a_{mk}
M	*Centroid*			

Table 1: Centroid

In Table 1 , the centroid $(1 \times k)$ is the average of all the input training vectors of the dimension $M \times k$, and forms the initial codebook.

- The master allocates the training vectors equally among all the slaves. The number of vectors to each slave are $\left\lfloor \frac{M}{P} \right\rfloor$
- $D-1$, which is the distortion value, is initialized to a very high positive value.
- The threshold € decides the termination condition of the algorithm.
- The splitting parameter 'δ' is initialized to a constant value.

Codebook Splitting:

The master doubles the size of the initial codebook. The increase in the number of codevectors is done by getting two new values, by adding and subtracting δ (which may be considered a variance, and is constant throughout the algorithm), to each centroid value. Therefore, the codebook splitting step generates a new codebook which is twice the size of the initial codebook.

$$[Centroid + \delta], [Centroid - \delta]$$

The codebook so formed in this splitting step is duplicated into the primary memory of all the slave processors.

Cell Allocation:

Each slave calculates the Euclidean distance between the input vectors allocated to it and the codebook vectors. Based on the Euclidean distance, it finds the nearest codebook vector to each input vector and allocates the particular input vector to the codebook vector.

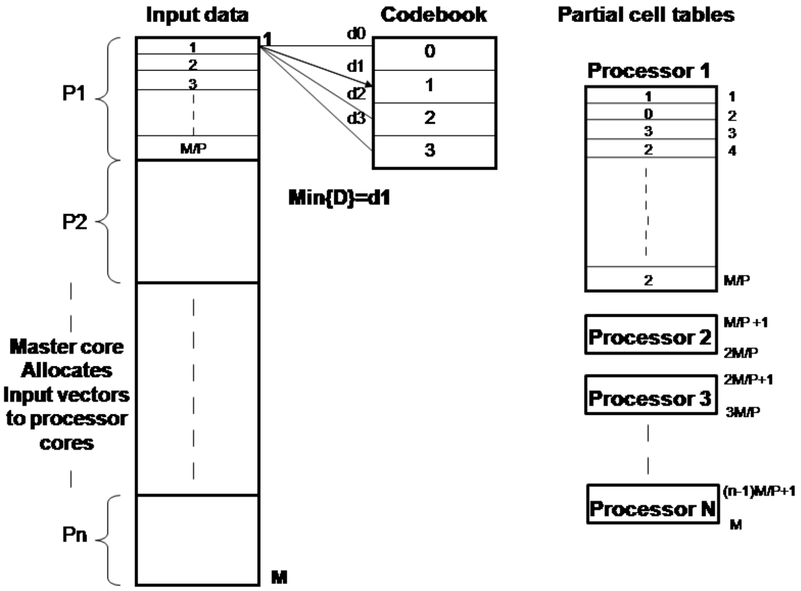

Figure 1: Input data allocation to processors and nearest codebook, cell table

In Figure 1, the initial training data is divided equally among all the processors

$$P = \{P1 \ldots Pn\}.$$

$$processor1 \leftarrow \left[1\ to\ \frac{M}{P}\right], \quad processor2 \leftarrow \left[\left(\frac{M}{P}+1\right) to\ 2M\right], \ldots processorN \leftarrow \left[\left(\frac{(n-1)M}{P}+1\right) to\ M\right].$$

Whenever a new codebook is formed, codevectors are duplicated into primary cache $(L1)$ of all slave processors. The Euclidean distance between each input vector allocated to the processor and all the codevectors is computed, minimum of all is taken and added to $'DPi(Distortion)'$. And the index of the codevector nearest to the input is placed in the corresponding location in the 'partial cell table' $\left(\frac{M}{P} \times k\right)$. Finally when all the allocated training vectors are computed, the 'partial cell table' and 'distortion' have to be communicated to the master, and this process is done by every other processor executing in parallel.

For each slave processor, the distortion value is $Di = \sum min\{D\}$ where $D = \{d1,\ldots,dr\}$ is the set of Euclidean distortions for each input vector with respect to the codevectors. And the corresponding 'distortion' values and 'partial cell tables' will be communicated to the master by all the slaves, and the master computes the 'total distortion' TDi and also integrates the individual cell table to form a 'final cell table' which is used for updating codebook.

The total distortion computed by the master is

$$TDi = \sum\{Di\}$$

The threshold value € which is initialized previously to a constant value is used to identify the termination of the algorithm. The termination condition for the algorithm is,

$$\frac{(\mathrm{TD_i} - 1 - \mathrm{TD_i})}{(\mathrm{TD_i} - 1)} \leq €$$

If the condition is satisfied, it implies that the optimum codebook for that level is formed for the given input training data set. And if the number of codevectors formed in that state is equal to the size of the codebook specified, terminate the program. Else go back to the Codebook Splitting step and proceed with the same.

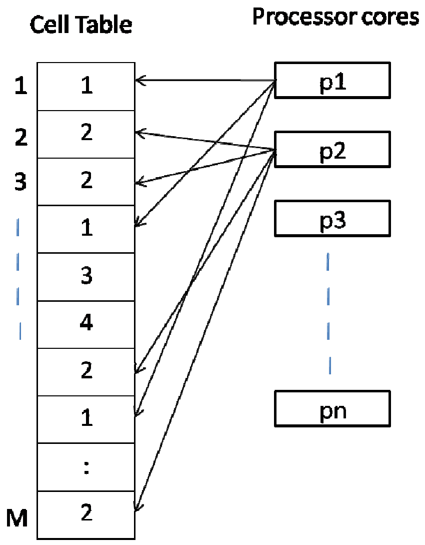

Figure 2: Parallel updation of codebook by slave processors

Codebook Updation:

If the Threshold condition is not satisfied then it implies that the codebook so formed is not the optimum one and needs to be updated. The codebook is updated by replacing each codevector by the centroid of all the input vectors allocated to it. The new centroids are calculated in parallel wherein with set of input vectors corresponding to one codevector are computed by a single processor. This procedure is explained from Figure 2.

In Figure 2, the cell table contains the indexes of codevectors to which the corresponding input vector is the nearest. A single processor updates single codevector at any point. The process of updation is:

- Extract all the input vectors which have the value $\mathbf{0}$ in the cell table. These are the set of input vectors which are nearest to codevector1.
- Compute the centroid of the vectors which have been extracted. This forms the new codevector.

Hence, if we have P slave processors available, then at any point of time P codevectors will be updated in parallel, and then the next round of updation proceeds and so on up to the number of codevectors, which executes in a round robin fashion. And the assigning of P codevectors to the P processors can be done randomly or serially. In the case of serial updation, if the size of the codebook in that stage is $'S'$, then S number of codevectors must be updated. If $CodevectorIndex \% P == \varphi$, implies processor φ performs the updation of that codevector. Once all the codevectors are updated go back to Cell Allocation step and continue the iterative procedure until the required numbers of codevectors are generated.

Performance

From Algorithm 1 described in the Section $\mathbf{3}$ it can be observed that the parallel processing tasks are identified separately. And from our experimental analysis, in the sequential version of Vector quantization using LBG algorithm, these parts of the program which can be parallelized were extracted out as separate process blocks (functions). And using a GNU gprof profiler, which helps in knowing where the program has spent most of its time and which process is called by which other process while it is executing, it is observed that parallel blocks consume $80\ to\ 85\%$ of the overall time and also provides the time consumed by individual parallel blocks. This parallelization can be accounted to two of the main processes in the algorithm.

1. Generation of 'Cell Table' also called as 'Allocation table,' i.e., allocating the input training vector to the nearest codebook vector based on Euclidean distance, which is a highly time consuming computation.

2. Calculation of Centroids in the Updation of codebook step.

The first step is the most time-consuming part and takes about $\mathbf{70}\ to\ 75\mathbf{\%}$ time of the total sequential part. In this step the entire input training data is divided into number of chunks equal to number of processors and allocated to them. As each processor has its own duplicated version of the codebook, further computations are done in parallel by the processors. Hence, the theoretical efficiency of the multi-core system or to say processor utilization' would be $100\%.$ The second step, which is calculation of centroids in the updation step, takes about $\mathbf{10}\ to\ 15\%$ of the sequential part. According to Amdahl's law:

$$speedup = \frac{1}{\left(S + \frac{(1-S)}{n}\right)}$$

Where S is time spent in executing the serial part of the parallelized version and n is the number of parallel processors. In the proposed algorithm, 85% of it can be parallelized, hence the time spent in executing serial part is 15% and assuming $n\ = \mathbf{4}$, the speedup of the parallel version is

$$\frac{1}{0.15 + \frac{1\ -\ 0.15}{4}} = 2.76$$

i.e., a quad-core system with has 4 cores would work $\mathbf{2.76}$ times as fast as the single processor system.

Results and Simulation

The proposed algorithm has been simulated using OpenMP with the training input size of 2000 vectors with the dimension varying from $2\ to\ 10$ .These results have been compared with the standard sequential execution and the results have been plotted. The graph clearly indicates the expected reduction in time consumed to create the codebooks.

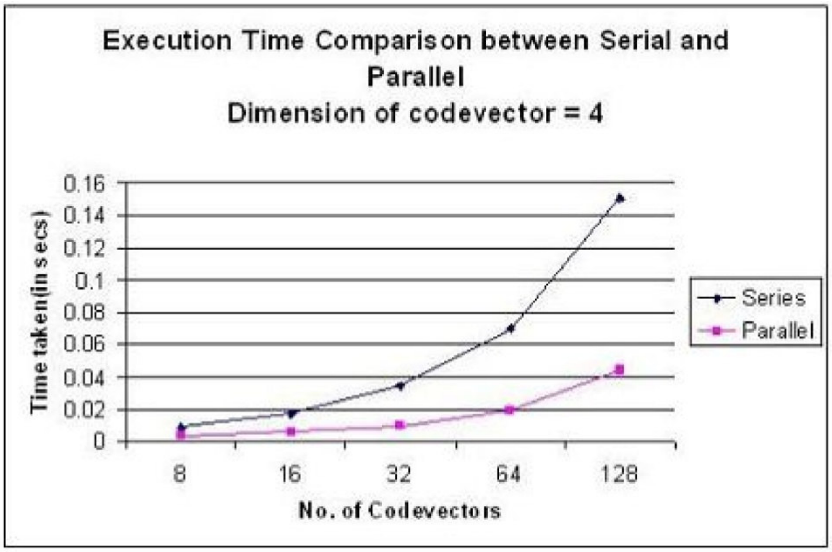

Figure 3: Execution Time Comparison

Figure 3 is the graph of the number of codevectors vs time taken to create the codebook. The number of processors is fixed at 4 and the number of codevectors is varied from $8\ to\ 128$. The results are plotted both for sequential as well as parallel algorithm. The difference is clear when the size of the codebook increases.

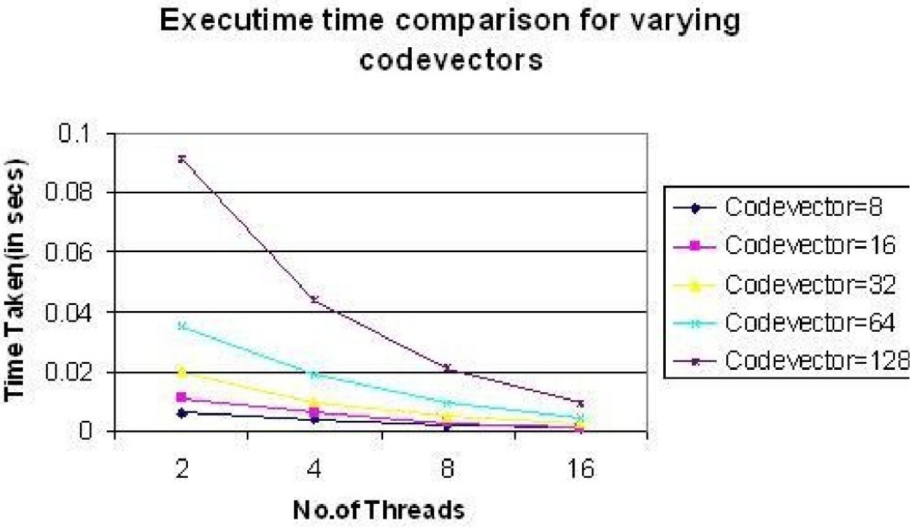

Figure 4: Plot of Time Taken vs. Number of threads

3 CONCLUSION & FUTURE WORK

In this paper a parallel implementation of LBG Vector quantization is demonstrated. The algorithm proposed here is general in the sense that it can be applied on any shared memory architecture irrespective of the underlying inter process communication. The performance of the proposed algorithm is analyzed and also experimentally simulated using the OpenMP shared programming. When compared with the sequential version of the LBG, the proposed algorithm has better performance in terms of speed of execution, and the study of execution time is done for varying number of parallel processing units.

The results obtained are the output of simulations of the sequential version of VQ using OpenMP. Implementation of the same using a multi-core system would provide accurate results regarding aspects like 'memory used' and 'core utilization' which were difficult to obtain in the present scenario. In a typical low-bandwidth network, consider a server which provides videos on demand. If the number of requests to the server is very high then the server sends a compressed version of the original high quality video. In this kind of scenario, the algorithm proposed here would greatly enhance the server response time by decreasing the time taken for video compression techniques which make use of LBG algorithm.

4 REFERENCES

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

[2] Toshiyuki Nozawa, Makoto Imai, Masanori Fujibayashi,and Tadahiro Ohmi: *A Parallel Vector Quantization Processor Featuring an Efficient Search Algorithm for Real-time Motion Picture Compression* ASP-DAC 2001: 25-26

[3] Akiyoshi Wakatani: *A VQ compression algorithm for a multiprocessor system with a global sort collective function*, data compression Conference .DCC 2006 proceedings.

[4] Troy Maurice Thomas: *Vector Quantization of Color Images Using Distributed Multiprocessors*, T.R. #USUEE-88-15, Utah State University, Logan, Ut., 1988.

[5] *Parallel codebook design for vector quantization on a message passing MIMD architecture,* Hazem M. Abbas, Mohamed M. Bayoumi 2002.

[6] Lee, H.J. Liu, J.C. Chan, A.K. Chui, C.K: *A parallel vector quantization algorithm for SIMD multi-processor systems,* Data Compression Conference, 1995. DCC '95 Proceedings

[7] Vandana S.Rungta: *parallel vector quantization codebook generation,* Utah State University, Logan, Utah 1991.

[8] Edward A. Fox: *DVI Parallel Image Compression*, Communications of the ACM, Vol 32, Number 7, July 1989, pp 844-851.

[9] C. Huang: *Large vector quantization codebook generation analysis and design*, Ph.D. dissertation, Utah State Univ., 1990

[10] W. H. Equitz: *A vector quantization clustering algorithm*, IEEE. Trans. ASSP. vol. 37, no. 10, pp. 1568-1575, Oct. 1989.