Optimal VLSI Architecture for Vector Quantization

Yu Hen Hu

University of Wisconsin
Department of Electrical and Computer Engineering
1415 Johnson drive
Madison, WI 53706
hu@engr.wisc.edu

Abstract

Optimal VLSI array structure design for the implementation of vector quantization (VQ) are investigated in this paper. After a brief review of the VQ algorithms, the algorithm and architecture design issues will be discussed. This is followed by a brief survey existing VQ implementation strategies and architecture.

I. VQ Algorithms

Given a feature vector v, a code book consisting of M code words $\{w(i);\ 1 \le i \le M\}$ and a distance (distortion) measure d(v, w(i)), the purpose of VQ is to find a code word index i* such that the corresponding distance d(v, $w(i^*)) = \min_{1 \le i \le M} d(v, w(i))$. For real time signal compression, a stream of feature vectors $\{v(n);\ n = 1, 2, \cdots\}$ need to be encoded at a rate no smaller than the rate these vectors are sampled. To achieve this goal, special purpose VSLI array structure may be needed.

In order to devise the code book, a set of training vectors$\{v(k);\ 1 \le k \le K\}$ will be used. The objective is to find M code words such that the average distortion

$$D = \frac{1}{K} \sum_{i=1}^{M} \sum_{k=1}^{K} I(v(k),i) d(v(k), w(i)) \qquad (1)$$

is minimized. In (1), I(v(k),i) is an *indicator membership function* defined by $I(v(k),i) = 1$ if $d(v(k), w(i)) < d(v(k), w(j))$, $j \neq i$; and $= 0$ otherwise. Sometimes the code book needs to be derived only once. More often, the code book needs to be updated from time to time. In either case, the training process of the code book often takes excessive computations when the code book size M and training set size K become large. Hardware assisted training will be desirable for certain applications.

All these factors have motivated researchers in the VQ area to study the most efficient hardware implementation strategy of the VQ algorithm. The most popular approach has been to use systolic VLSI array processors due to the inherent parallelism in code book search [1, 3, 4, 8, 9, 12, 13, 14, 15, 16, 17]. Recently, there are also a number of reports on adaptive vector quantization implementation using mainly neural networks [2, 5]. These existing results provide a rich collection of ad hoc VQ architecture. For a given set of specific performance requirements and resource constraints, a designer still has to decide whether to use an existing approach, or to devise a new architecture for the problem on hand.

II. VQ Hardware Design Issues

A typical VQ design specification would contain the following components:

Performance specification – Given a reasonably sized training and testing file (e.g. 60000 training vectors, 8000 testing vectors), devise a VQ method (by selecting a specific type of VQ, a feature extraction method, the feature dimension, code book size, and a distortion measure) such that both the real time processing requirement (e.g. 20 ms/frame at 8KHz sampling rate, 10 LSF coefficient per frame), and VQ performance requirement (e.g. 24 bits/frame, SNR > 20 dB, or other perceptual based performance measures) are met.

Resources specification – Design a special purpose VQ VLSI array structure which implements the specified algorithm to achieve the performance requirements, under some physical design constraints (e.g.. maximum physical size, maximum power consumption, maximum I/O band-width, etc.), while minimizing the implementation and maintenance cost.

The most crucial design decisions to be made during a VQ processor design are (i) to select the particular VQ algorithms, be it full search VQ, tree-based VQ, split-VQ, or cascaded VQ, and (ii) to determine code book updating policy (e.g.. fixed code book versus various adaptive VQ schemes). As of now, these high-level design decisions are often reached based on past experiences, and designer's personal judgment. To our knowledge, no systematic design methodology is available to aid this decision process. Once these major design decisions are made, a number of derivative design decisions will need to be made. These secondary algorithm related design decisions include feature extraction, distortion measure selection, code word dimension (N) and the code book size (K).

0-7803-2431-5/95 $4.00

In designing the VLSI array structure, it is necessary to determine what kind of processing unit to use. If off-the-shelf processing units are used, they should have special hardware such as multiple communication ports to support pipelined VLSI array processing. Examples of this type processors include TMS32040®, and Transputer®. The other option is to design special purpose hardware. The choice of processing unit has profound implications on subsequent array design: First, its inter-processor communication capability will affect the preferred array configuration, as well as the *granularity* used in parallel processing. Secondly, its ability to address local memory may affect the way code book is distributed in the processor array. Clearly, other physical features such as packaging size, power consumption will also affect the overall system.

Once the processing unit is decided, subsequent design decisions are needed for array structure design. These design decisions include array configuration (e.g.. linear array, 2D array, hexagonal array, etc.), number of processing units, data input/output and internal data movement patterns, as well as local memory organization. Among these decisions, the array configuration is intimately related to how the algorithm is *mapped* to the array structure. This brings up the issue of algorithm-architecture interaction.

III. Algorithm Architecture Interaction

Mapping Algorithm to Array Structure – A well-developed VLSI array structure design methodology is to map a *localized*, regular, iterative algorithm (RIA), represented by a regular, shift-invariant data dependence graph, to a locally connected, pipelined array structure [10]. Often, a given computing algorithm formulation needs to be transformed into the desired localized RIA format in order to exploit maximum concurrence. The array configuration, in this design style, is largely determined by an affine transformation from the algorithmic index space to the processor index space. On the other hand, more sophisticated design methodologies have been developed recently to meet the architectural constraints. For example, when consecutive RIA loops are to share the same processor array, a *multi-stage systolic mapping* method [7] can be used to better interface operations in subsequent loops. If the array size is limited due to a constraint on the number of processing units, partitioning methods have been proposed to address this problem [6].

Real time processing constraint – The need to process VQ encoding at a rate which matches the signal processing throughput rate imposes great constraints on both VQ algorithm design as well as architectural design. Generally speaking, high speed processing implies less complicated algorithm, and even inferior performance. However, with VLSI array structure, high speed can also be achieved using more hardware provided the algorithm is *scalable*.

Storage Limitation – Various VQ algorithms are derived primarily to contain the exponentially growing code book size of a full-search VQ. They achieve this goal by trading VQ efficiency to smaller code book size. However, when VLSI array structure is used, smaller code book size alone does not necessarily imply faster throughput rate. Regularity, pipelinability will be additional factors which determines the achievable throughput. The choice of processing units also affects the maximum size of local memory, and hence the storage limitations.

VI. Full Search VQ Array Structure Design

VI.A VLSI Architecture for VQ Encoding

The VQ encoding process can be re-formulated in a localized RIA format as follows:

VQ Encoding Algorithm
```
for n = 1, 2, • • •
   dmin = ∞,  i* = 0
    for i = 1 to M
       evaluate d(v(n),w(i))
       if  dmin > d(v(n),w(i)),
       then  dmin = d(v(n),w(i)), i* = i; end
   end % i-loop
end
```
If the evaluation of d(v, w(i)) takes T_0 time units, and the comparison of two distances takes T_c time units, them the total sequential computation time to encode a single date vector v will be $M \cdot T_0 + (M-1)T_c$ time units. Usually, $T_0 >> T_c$. Thus, roughly speaking, the time taken to encode a single code vector v on a sequential machine will be approximately MT_0 time units. If the incoming data throughput rate exceeds $1/(MT_0)$ data vectors per time unit, a single CPU will not be able to keep up with the throughput demand, and parallel processing must be used. Motivated by this observation, previously, various VLSI array structures using systolic array or wavefront array [10] have been proposed [3, 9, 18]. These results are based on a common observation, namely, the VQ encoding is an regular, iterative algorithm, and parallelism can readily be exploited.

We claim the VQ encoding is inherently parallel because the computation within the loop body is not dependent on computation in other iterations. Using the standard VLSI array linear projection design methodology [10], a linear array of M processors each responsible for the processing of the loop body of the i^{th} iteration of the i-loop can easily be devised.

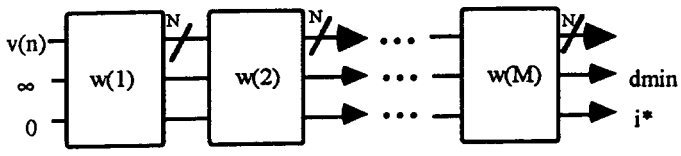

Figure 1. Linear VSLI Array VQ Encoder

As depicted in Figure 1, the sample vector v(n) propagates from left to right. Within each processor, the distortion d(v(n),w(i)) is evaluated, and compared to the quantity d_{min} available from the neighboring processor on the left. If d(v(n),w(i)) < d_{min}, then dmin and i* will be updated as d_{min} = d(v(n),w(i)), and i* = i. Otherwise, they will remain unchanged. Upon completion, the input feature vector v(n), d_{min} and i* will be propagated to the processor to the right, continuing the execution of the present iteration. While v(n), and corresponding dmin, i* are being processed in processor #2, the processing of the new sample v(n+1) can be initiated in processor #1. As a result, roughly every T_0 time units, a new feature vector v(n) can be processed. That is, the effective throughput rate is $1/T_0$ one sample per time unit. This represent a linear speedup factor of M compared to the single processor implementation.

With the linear array implementation, the number of processors is the same as the number of code words and each processing unit stores exactly one code word. If the demand of throughput rate is much smaller than $1/T_0$ but larger than $1/(MT_0)$, one may group adjacent processing units into a single processing unit. If g processing units are replaced by a single processing unit with g times storage space (i.e. storing g code words), the total number of processing units can be reduced to M/g, while the throughput rate also reduced to $1/(gT_0)$. However, the latency remains unchanged.

On the other hand, if the desired throughput rate is higher than $1/T_0$, several linear arrays of processors as shown in figure 1 can be used in an interleaved fashion to meet the demand. For example, in Figure 2, we illustrate interleaving three linear arrays to enhance the throughput by three folds.

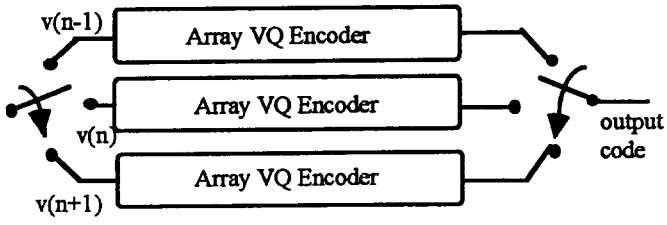

Figure 2. Interleaved Array VQ encoder

V.B. VLSI Array for Code Book Training

A most popular VQ code book training algorithm is the *k-means* algorithm, sometimes also known as the *LBG* algorithm [11]. We will show that with proper algorithm transformation, the k-means algorithm can also be implemented with the same linear processor array as used in the VQ encoding with little overhead:

A fringe benefit of using a linear array structure(Figure 1) is that it can easily be modified to implement the k-means training algorithm. Note that during the VQ encoding phase, the minimum distortions $d_{min} = \min_i d(w(i), v(k))$ will be evaluated for each training vector v(k). Hence the total distortion $D = \sum_k d_{min}$ can be evaluated with simple summation operations. Moreover, since every training sample has to traverse through the entire linear array, each processing unit can place the vector v(k) in a temporary storage space. Once d_{min} is found, the i*-th processing unit such that I(v(k),i*) = 1 can be informed to update its new code word with v(k). Incorporating these changes into the VQ encoding algorithm, we have the following implementation of the k-means training algorithm using a slightly modified linear array of processors:

k-Means Clustering Algorithm

```
Converged = FALSE
Repeat until converged,
   D(0)=0;  wnew(i) = 0, count(i) = 0, 1 ≤ i ≤ M.
   for k = 1 to K,
      dmin(k,0) = ∞,  i*(k,0) = 0
      for i = 1 to M
         evaluate d(w(i),v(k))
         tmp(i) = v(k)
         if   dmin(k,i–1) > d(w(i),v(k)),
         then dmin(k,i) = d(w(i),v(k)),   i*(k,i) = i;
         else dmin(k,i) = dmin(k,i–1),
                  i*(k,i) = i*(k,i–1)
         end % if
      end % i-loop
      D(k)=D(k–1) + dmin(k,M)
      index(M+1)=i*(k,M)
      for i = M to 1,
         index(i) = index(i+1)
         if i = index(M), then
            wnew(i) = wnew(i) + tmp(i)
            count(i) = count(i)+1
         end % if
      end % i-loop
   end % for k loop
   for i = 1 to M,
```

```
    w(i) = wnew(i)/count(i);
  end
  if 1–Dold/D(K) < ε, then converged = TRUE
  else Dold = D(K).
end % repeat loop
```

Below is a modified dual-linear processor array for the implementation of the k-means training algorithm. Note that new code words {w(i); $1 \le i \le M$} is computed only once for each presentation of the K training samples.

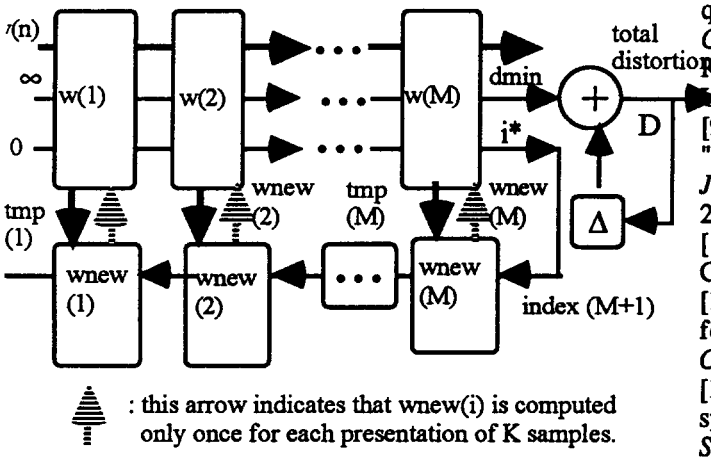

Figure 3. A dual linear array which computes new weights, and total distortion

VI. Conclusion

Due to space limitation, we are unable to present VLSI array structure implementation of other VQ algorithms. A full report will be available at the time this paper appears.

REFERENCE

[1] Chen, O. T.-C., B.J. Sheu, and Zhen Zhang, "An adaptive vector quantizer based on the Gold-Washing method for image compression," *IEEE Trans. on Video Technology*, vol. 4, no. 2, pp. 143-157, 1994.

[2] Chen, O. T.-C., B. J. Sheu, and W.-C. Fang,, "Image compression using self-organization networks,"*IEEE. Trans. onVideo Technology*,vol.4, no.5,pp. 480-489, 1994.

[3] Davidson, G. A., P. R. Cappello, and A. Gersho, "Systolic architectures for vector quantization," *IEEE. Trans. on ASSP*, vol. 36, no. 10, pp. 1651-1664, 1988.

[4] Fang, W.-C., Chi-Yung Chang, B.J. Sheu, O.T.-C. Chen, and J.C. Curlander, "VLSI systolic binary tree-searched vector quantizer for image compression," *IEEE Trans. on VLSI Systems*, vol. 2, no. 1, pp. 33-44, 1994.

[5] Fang, W. C., B. J. Sheu, and O.T.-C. Chen, "A neural network based VLSI vector quantizer for real-time image compression," in Proc. *DCC '91. Data Compression Conference*, Snowbird, UT, USA, J. A. Storer and J.H. Reif, Ed., IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1991, pp. 342-351.

[6] Hwang, Y.-T., and Y. H. Hu, "A unified partitioning and scheduling scheme for mapping multi-stage nested DO loop programs onto a distributed memory system," *J. of VLSI Signal Processing*, no. pp. (to appear), 1994.

[7] Hwang, Y. T., and Y. H. Hu, "MSSM - A design aid for multi-stage systolic mapping," *J. of VLSI Signal Proc.*, vol. 4, no. 2/3, pp. 125-146, 1992.

[8] Israelsen, P., "VLSI implementation of a vector quantization processor," in Proc. *Data Compression Conference* , Snowbird, UT, USA, J. A. Storer and J. H. Reif, Ed., IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1991, pp. 463.

[9] Kolagotla, R. K., Shu-Sun Yu, and J.F. Jaja, "Systolic architectures for finite-state vector quantization," *Journal of VLSI Signal Processing*, vol. 5, no. 2-3, pp. 249-259, 1993.

[10] Kung, S. Y., *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[11] Linde, Y., A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE. Trans. on Communications*, vol. 28, no. 1, pp. 84-95, 1980.

[12] McCanny, J. V., and M. Yan, "A VQ tree search system based on bit level systolic arrays," in Proc. *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, vol. 2, Maple Press, San Jose, CA, USA, 1989, pp. 743-747.

[13] Panchanathan, S., and M. Goldberg, "A systolic array architecture for image coding using vector quantization," in Proc. *International Symposium on VLSI Technology, Systems and Applications*, Taipei, Taiwan, ROC, IEEE, 1989, pp. 271-275.

[14] Park, H., and V.K.P. Kumar, "Modular VLSI architectures for real-time vector quantization," in Proc. *IEEE ISCAS*, Singapore, vol. 1, IEEE, NY, NY, USA, 1991, pp. 188-191.

[15] Qureshi, Q. A., and T. Fischer, "A hardware processor for implementing the pyramid vector quantizer," *IEEE Trans. ASSP*, vol. 37, no. 7, pp. 1135-1142, 1989.

[16] Tsang, K., and Belle W. Y. Wei, "A VLSI architecture for a real-time code book generator and encoder fo a vector quantizer," *IEEE Trans. on VLSI Systems*, vol. 2, no. 3, pp. 360-364, 1994.

[17] Yan, M., and J. V. McCanny, "A bit-level systolic architecture for implementing a VQ tree search," *Journal of VLSI Signal Processing*, vol. 2, no. 3, pp. 149-158, 1990.

[18] Yan, M., J. V. McCanny, J.V., and Y. Hu, "VLSI architectures for digital image coding," in Proc. *ICASSP 90. 1990 ICASSP*, Albuquerque, NM, USA, vol. 2, IEEE, New York, NY, USA, 1990, pp. 913-916.