Journal of VLSI Signal Processing 28, 235–243, 2001

Scalable Linear Array Architecture with Data-Driven Control for Ultrahigh-Speed Vector Quantization

DING-MING KWAI AND BEHROOZ PARHAMI
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

Received October 1999; Revised April 2000

Abstract. Current and future requirements for adaptive real-time image compression challenge even the capabilities of highly parallel realizations in terms of hardware performance. Previously proposed linear array structures for full-search vector quantization do not offer scalability and adaptivity in this context, because they require separate data/control pins for dynamically updating the codevectors and complicated interlock mechanisms to ensure that the regular data flow is not corrupted as a result of updates. We explore the design space for full-search vector quantizers and propose a novel linear processor array architecture in which global wiring is limited to clock and power supply distribution, thus allowing high-speed processing in spite of only limited communication with the host via the boundary processors. The resulting fully pipelined design is not only area-efficient for VLSI implementation but is also readily scalable and offers extremely high performance.

Keywords: control scalability, data-driven control, image compression, pipelined implementation, systolic array, vector quantization, VLSI architecture

I. Introduction

Data compression allows the transmission and storage of large amounts of data using modest hardware resources (communication bandwidth and memory). Even though at inception, data compression was motivated by the limited bandwidth of data transmission facilities, reliance on compression techniques has continued despite phenomenal growth in the available bandwidth. Besides improved performance of compression algorithms and associated hardware, there are several other reasons for the continued use of data compression. While communication links now offer vastly improved bandwidths, data transmission is still limited at the end points and at intermediate switches by logic circuit latency and buffer capacity. Furthermore, data compression leads to fewer access conflicts and/or greater concurrency when high-capacity resources are shared among multiple users.

Vector quantization (VQ) has been applied to data compression in certain speech and image processing applications [1–6]. A VQ system parses the input data stream into nonoverlapped sequences or vectors. According to Shannon's source coding theory, better performance can be achieved by coding vectors rather than scalars. The basic computation associated with VQ is a pattern matching process, where each input vector (pattern) is compared to a finite set of representative codevectors (templates) to identify a closest match. The codevectors are contained and indexed in a codebook that allows the indices of matching codevectors to be transmitted or stored in lieu of the input vectors. These indices are later used to recover the corresponding codebook entries.

The computational complexity of VQ-based systems is primarily associated with the encoding process, since the reverse decoding process is a relatively simple table lookup. The complexity of the encoding process arises from the need to search through the entire

codebook. Adopting tree search, with the codebook organized in multiple levels, significantly reduces the computational complexity at the expense of increased storage requirements and degraded quality in terms of the signal-to-noise ratio [4]. Because an L-level tree-search VQ encoder can be realized by means of a cascaded series of L full-search VQ encoders [7], we will focus on the design of full-search VQ encoders.

Scalability and realizability are among the most important attributes of parallel architectures [8]. Aspects of scalability include bounded I/O requirements [9] and feasibility of global clocking in the presence of propagation delays [10]. Realizability is aided by modularity, control simplicity, and low interconnection density. For these reasons, most designs for parallel VQ encoders are based on linear array architectures [1, 11–14].

However, previously proposed designs tend to treat the memory access requirement as an independent issue, implying that load and store operations needed for dynamic or in-operation updating of codevectors, essential for many applications [2–5], are performed via separate I/O pins or else codevectors are fixed at the time of manufacture. With existing designs, this capability is in conflict with the requirements of modularity and scalability that dictate channeling all communications with the host through the boundary processors in the linear array. Our data-driven control scheme [15] allows us to include these desirable features in the linear array, while enhancing the performance of its pipelined and parallel operation.

The rest of this paper is organized as follows. Section II briefly describes the VQ full-search algorithm. Section III introduces a unidirectional linear array, along with its processor structure, for VQ encoding. Several issues not addressed in [16] are covered here. In Section IV, we propose a data-driven control scheme to solve the problems of data loading, array initialization, and index labeling. Section V examines various design choices for the processor structure. In Section VI, these designs are compared with regard to cost and performance. Section VII contains our conclusions.

II. VQ Full-Search Algorithm

Let x_k be an M-dimensional vector, where the subscript k indicates its position in a stream of source vectors entering a VQ encoder. The VQ encoder searches through the codebook $\{w_i\colon 0 \le i \le N-1\}$ and derives an index I_k pointing to the best matched codevectors for each input vector x_k. Associated with the pattern matching process is a measure of distortion, which serves as a basis for quantifying the dissimilarity between each codevector w_i and the input vector x_k. The squared error distortion measure between $x_k = (x_{k0}, x_{k1}, \ldots, x_{k,M-1})$ and $w_i = (w_{i0}, w_{i1}, \ldots, w_{i,M-1})$ is often used, primarily because of its mathematical simplicity. The squared error distortion measure can be formulated as follows:

$$\|x_k - w_i\|^2 = \sum_{j=0}^{M-1} \left(w_{ij}^2 - 2w_{ij}x_{kj} + x_{kj}^2\right)$$

For the purpose of codebook search, the sum over j of ${x_{kj}}^2$ in the equation above is common to all codevectors and hence will not affect the comparison result. This simplifies the distortion measure to an inner-product form of M multiplication-accumulation steps.

$$d(x_k, w_i) = \sum_{j=0}^{M-1} w_{ij}(w_{ij} - 2x_{kj})$$

The computation is actually very similar to a matrix-vector multiplication [17, 18]. In addition, the VQ encoder needs to find the minimum among all the N distortion measures. The index $I_k = i$ is selected if $d(x_k, w_i)$ yields a minimal value d_k. We describe the full-search VQ in the following regular iterative algorithm. Here and throughout, $+\infty$ ($-\infty$) denotes the largest positive (smallest negative) number that can be represented with the PE's arithmetic.

Algorithm 1: Full-search VQ
for $0 \le i \le N-1$
  for $0 \le j \le M-1$
    if $i = 0$ or $j = 0$
    then ${x_{kj}}^{(-1)} \leftarrow x_{kj}$, ${y_{ik}}^{(-1)} \leftarrow 0$
    endif
    ${x_{kj}}^{(i)} \leftarrow {x_{kj}}^{(i-1)}$,
    ${y_{ik}}^{(j)} \leftarrow {y_{ik}}^{(j-1)} + w_{ij}(w_{ij} - 2{x_{kj}}^{(i-1)})$
  endfor
  ${d_{kM}}^{(-1)} \leftarrow +\infty$, ${I_{kM}}^{(-1)} \leftarrow 0$
  if ${y_{ik}}^{(M-1)} < {d_{kM}}^{(i-1)}$
  then ${d_{kM}}^{(i)} \leftarrow {y_{ik}}^{(M-1)}$, ${I_{kM}}^{(i)} \leftarrow i$
  else ${d_{kM}}^{(i)} \leftarrow {d_{kM}}^{(i-1)}$, ${I_{kM}}^{(i)} \leftarrow {I_{kM}}^{(i-1)}$
  endif
endfor

Figure 1 depicts the dependence graph of the full-search VQ algorithm for $N = 4$ and $M = 3$. We have

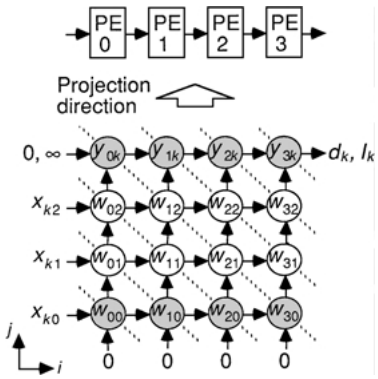

Figure 1. Dependence graph for full-search VQ and its mapping onto a linear array of PEs. Dotted diagonal lines represent the equitemporal lines of a time schedule for the linear array.

added the top row of shaded nodes to show the comparison steps following the inner-product computation in each column. Note that in the dependence graph, 0s are input to the bottom row of shaded nodes that begin the inner-product computations. These shaded nodes are thus different from the other nodes that receive intermediate results from the neighbors just below them in the respective columns. Note that the computation in the topmost row of Fig. 1 can be overlapped with the next computation, so that the input data stream is fed continuously to the array without any interspersed delay.

III. The Linear Array Architecture

Our design is modular; identical PEs are cascaded into a unidirectional linear array, as shown at the top of Fig. 1, and, with more detail, in Fig. 2. The architecture is derived by projecting the dependence graph along the j direction. The projection direction is coincident with the vector dimension so that the ith column of nodes map onto the ith PE [16]. The linear array has two types of channels: c_{kj} is for control signals, via which a 2-bit control pattern c_1c_2 passes through unchanged; x_{kj}, d_{kj}, and I_{kj} are for data signals of the input vector element, an intermediate result representing the minimal distortion thus far, and the index of the codevector yielding the latter minimal distortion, respectively.

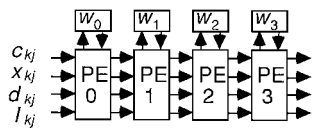

Figure 2. Linear array for VQ codebook full search.

The operation of the linear array is similar to that of a distributed pipeline [19]. The input data stream travels from left to right, at the rate of one PE per clock cycle. Each PE performs M inner-product steps with its stored codevector for each input vector that it receives element by element. Each element processed is also passed on to the right where matching against the next codebook entry is performed. The distortion measure is obtained by summing the products and comparing the result to the minimal value computed by all the preceding PEs. This latter value is an input to the PE from its left. This value and its associated index, or a smaller value obtained by the PE along with its index, are then passed to the right. The next PE repeats this process, and so on.

Figure 3 shows the detailed structure of each PE. It is composed of data latches (rectangular boxes), two way multiplexers and demultiplexers (circles with a cross), a comparator module (CMP), and a multiply-accumulate (MAC) module using appropriate sign extension to accommodate the required range of values. At the ith PE, the codevector w_i, which is stored in a circular queue [11, 20, 26] is recycled through the queue such that its element w_{ij} emerges as the element x_{kj} of the input vector x_k arrives. This allows the computation of w_{ij} $(w_{ij} - 2x_{kj})$ to be performed. The result is stored in an accumulator for combining with other products.

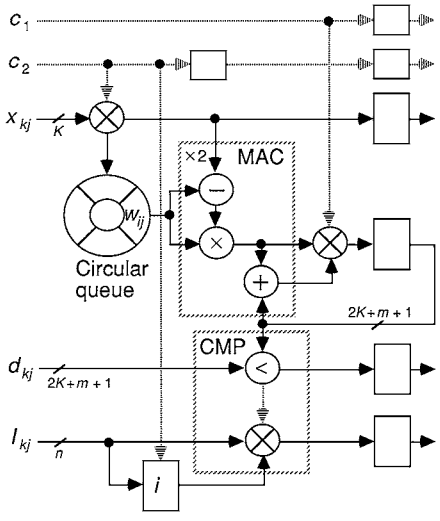

Figure 3. PE structure for full-search VQ implemented on a linear array. Dotted lines represent single-bit control paths and solid lines represent data words of varying widths ($n = \lceil \log_2 N \rceil$, $m = \lceil \log_2 M \rceil$).

Due to the rhythmic operation of VQ encoding, the circular queue can be implemented as a random-access memory (RAM), with a counter linked to the global clock supplying the address. The address counter is continuously incremented up to the length of the circular queue and then reset to zero. Using such a circular queue allows the loading of new codevectors to be pipelined with normal operation, thus wasting no clock cycle before or after, and also makes it possible to vary the vector dimension. This is in contrast to implementations with read-only memory (ROM), where the code vectors are of fixed dimension and are assigned fixed values during fabrication. To flexibly determine the vector dimension M, we have assumed that the circular queue is of variable length. In practice, the value of M is selected in the range from 16 to 36 [11]. Therefore, a 6-bit address counter is sufficient for such applications.

Note that with the circular queue and its counter, one can view the lowermost shaded nodes in the dependence graph of Fig. 1 as resetting the counter to zero and each regular node in the same column incrementing the counter value by 1. We can attach control tag bits to the data stream to distinguish between the different functions that a node can perform. In the next section, we will discuss this data-driven control scheme as related to the 2-bit "opcode" c_1c_2 shown in Fig. 3.

IV. Data-Driven Control Scheme

In our design, global wiring is limited to power and clock distribution. Because data and control signals are not broadcast but propagated exclusively through local connections, we need to devise a control scheme to allow the loading and storing of data to occur by means of a control bit pattern that is supplied to the boundary PE at the left end of the array [15]. The 2-bit control pattern c_1c_2, which is sufficient for specifying PE operations as data traverses the linear array, is coincident with each element x_{kj} in the input vector x_k. The resulting design is readily expandable through the inclusion of additional PEs to accommodate a larger codebook. For this to be possible, the index labels of the PEs must also be dynamically changeable.

The I/O pins for passing the data elements x_{kj} and the indices I_{kj} are also used for loading of new codevectors and new PE labels, respectively. Note that applying $-\infty$ to the d_{kj} input causes the incoming index I_{kj} to pass through unchanged. In this way, the latches holding I_{kj} form a shift register of length N (a partial scan path, in the terminology of built-in self-test) that allows us to set $x_{kj} = w_{ij}$ and $I_{kj} = i$, thus forcing the weight w_{ij} and the label i to be stored in the ith PE. The process above can be explained by the augmented dependence graph shown in Fig. 4. The added black nodes change the direction of data flow from horizontal to vertical (projection) direction, implying that from that point on, the data will remain stationary at the PE. Recall that we projected each column of the dependence graph onto one PE.

In Fig. 4, the triples $(x_{kj},\ d_{kj},\ I_{kj}) = (w_{ij}, -\infty,\ i)$, for $0 \le i \le N-1$ and $0 \le j \le M-1$, correspond to the data being sequentially inserted, in ascending order of the indices, during the initialization phase. Data elements are propagated along the array and are stored in the circular queue and the index register upon arrival at the intended PE. In general, the PE indices need not be consecutive on the physical array. If spare PEs and bypass connections are provided for the linear

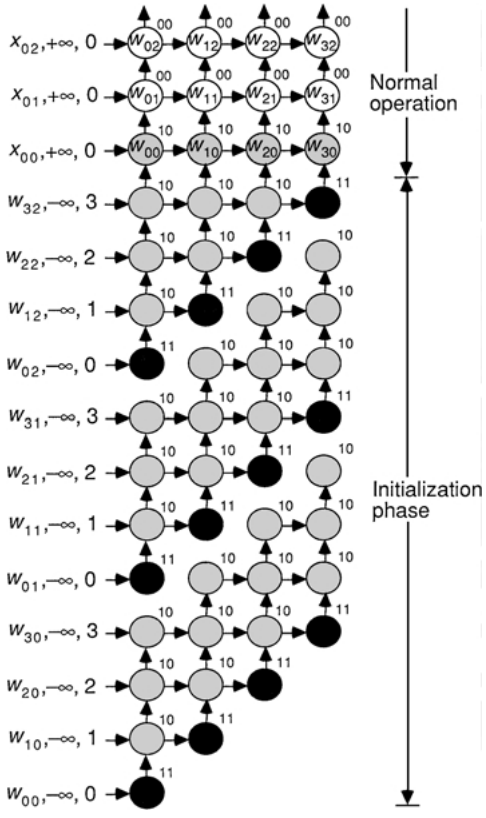

Figure 4. Augmented dependence graph that includes codevector load/store and index labeling operations.

array [8, 20], replacement of faulty PEs can be accomplished without modifying the computation scheme. Thus, extension of the design to offer fault tolerance is easily accomplished, particularly in view of the built-in partial scan paths that facilitate fault detection and diagnosis.

The control bit pattern $c_1c_2 \in \{00,\ 10,\ 11\}$, shown next to each node in Fig. 4, marks three distinct node types performing different functions. Observe that the bit stream c_1 flows in the horizontal direction, while c_2 flows in the diagonal direction. The latter corresponds to taking two clock cycles to pass through each PE; hence the need for two delay elements on the path of c_2 within each PE. From Fig. 4, it can be easily verified that the control bit pattern c_1c_2 correctly instructs the linear array to load/store the codevectors first and then to compute/compare the distortion measures. The function table relating the PE's operation to the control tag c_1c_2 is shown in Fig. 5.

In the initialization phase, c_2 takes on the value 1 followed by a string of $M-1$ 0s, with the pattern repeated N times for N codevectors. Throughout this period, c_1 remains 1. During normal operation, c_1 takes on the value 1 followed by a string of $M-1$ 0s, while c_2 remains 0. Note that the c_1 bit stream also indicates the time to retrieve the minimal distortion measure and the index of the corresponding codevector with a 0-to-1 transition at the output of the rightmost PE.

A complete description of the operation of the ith PE in the linear array is given in Algorithm 2, where the superscripts $(i-1)$ and $(j-1)$ denote the preceding PE and the previous clock cycle, respectively. Parts I, II, and III of Algorithm 2 are mutually exclusive and each is assumed to complete in a single clock cycle. The control mechanism is very simple, requiring only 2-way multiplexers/demultiplexers for its implementation (Fig. 3). There is an increase of 4 in the total pin count, two each for input and output. Compared to the number of data I/O pins, this constitutes a relatively small overhead.

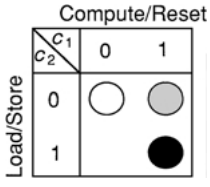

Figure 5. Node functions associated with the three possible control tag bit combinations.

Algorithm 2: Operation of the *ith* PE in full-search VQ encoding

//I. Data loading and index labeling (black node)
if $c_1c_2 = 11$
then $w_{ij} \leftarrow x_{kj}^{(i-1)}$, $i \leftarrow I_{kj}^{(i-1)}$, $M \leftarrow M+1$
endif

//II. Initialization and comparison (shaded node)
if $c_1c_2 = 10$
then $j \leftarrow 0$
$x_{kj}^{(i)} \leftarrow x_{kj}^{(i-1)}$,
$y_{ik}^{(j)} \leftarrow w_{ij}\ (w_{ij} - 2x_{kj}^{(i-1)})$
if $y_{ik}^{(j-1)} < d_{kj}^{(i-1)}$
then $d_{kj}^{(i)} \leftarrow y_{ik}^{(j-1)}$, $I_{kj}^{(i)} \leftarrow i$
else $d_{kj}^{(i)} \leftarrow d_{kj}^{(i-1)}$, $I_{kj}^{(i)} \leftarrow I_{kj}^{(i-1)}$
endif
endif

//III. Inner-product step (clear node)
if $c_1c_2 = 00$
then $j \leftarrow j+1$
$x_{kj}^{(i)} \leftarrow x_{kj}^{(i-1)}$,
$y_{ik}^{(j)} \leftarrow y_{ik}^{(j-1)} + w_{ij}\ (w_{ij} - 2x_{kj}^{(i-1)})$
endif

In Algorithm 2, incrementation of j within part III represents the advancing of the circular queue counter and incrementation of M in part I updates the counter's reset value for wraparound.

V. Serial/Parallel Processing Tradeoffs

The proposed array architecture for VQ encoding can be implemented in many different ways. In this section, we analyze the cost and performance of such arrays with four implementation alternatives. These alternatives are based on concurrency, or lack thereof, in handling words (word-parallel vs. word-serial) or bits within words (bit-parallel vs. bit-serial). The evaluation parameters include the number of PEs, circuit size, number of data I/O pins, throughput rate, and pipeline latency. Table 1 summarizes the results.

The circuit size per PE of each design is estimated based on a gate-level analysis of the various components. The contribution of the codebook memory is not considered, because its total size is MNK bits for each of the four designs. Ripple-carry addition and array multiplication are assumed for the multiply-accumulate module [21, 22]. For bit-serial designs, multiplications are done by sequences of shift-add operations, beginning with the least-significant bit (LSB).

Table 1. Comparison of various array designs for VQ encoding (N = codebook size, M = vector dimension, K = word width, $n = \lceil \log_2 N \rceil$, $m = \lceil \log_2 M \rceil$).

	Word-parallel, bit-parallel [11]	Word-parallel, bit-serial [14]	Word-serial, bit-parallel	Word-serial, bit-serial [12]
Number of PEs	N	N	N	MN
Circuit size (gates/PE)	$42MK(2K+m)$	$26M(K+2)+16m$	$10K(K+8)+48m$	$42K+27m$
Data I/O (pins/PE)	$2(MK+2K+m+n+1)$	$2(M+2)$	$2(3K+m+n+1)$	18
Throughput (samples/cycle)	$M(2K+m+1)$	M	1	$1/(2K+m+1)$
Pipeline latency (clock cycles)	$K(M+N)$	$N(2K+m+1)+2K+M$	$M+N$	$MN(2K+m+1)$

With an initial word width K, the 2's-complement representation of $w_{ij} - 2x_{kj}$ requires $K+2$ bits. Multiplying the difference by w_{ij} leads to a $(2K+1)$-bit number and accumulating M such products requires an added width of $m = \lceil \log_2 M \rceil$ bits if overflow error is to be avoided. The total number of bits for computing and passing the minimal distortion value is thus $2K+m+1$. Here, we assume that the added range discussed above is in fact required for proper operation. For bit-serial designs, the input data has to be widened with sign extension, thus reducing the throughput rate by a factor $1/(2K+m+1)$.

The word-parallel, bit-parallel design [11] can be viewed as a direct implementation of Fig. 1. Such a design requires a large number of I/O pins; for example, there will be 312 I/O pins for $N=256$, $M=16$, and $K=8$. Considerably fewer I/O pins are needed in the word-parallel, bit-serial design. Note that the longest propagation time is dominated by the $(2K+m+1)$-bit addition. Both of the word-parallel designs given in [11, 14] achieve a speedup of $2K+m+1$ compared to the respective word-serial designs by using a shorter clock cycle and operating on a bit-by-bit basis. However, alignment of the incoming and outgoing data signals requires the insertion of a large number of latches. These skewing and deskewing delays may lead to difficulties in concurrent loading of codevectors. These designs do not specify a control scheme. The suggestion that a global data bus with broadcast control can be used to solve the problem [18] is undesirable for scalability reasons.

Unlike the designs above in which the codebook is distributed throughout the array, the word-serial, bit-serial design [8] utilizes an external centralized memory to store the codebook. Several pairs of channels are used for transporting data that travel at two different speeds. The design has the distinguishing feature of constant I/O bandwidth, provided that the precision of the distortion value is less than M bits. It is worth mentioning that the design also incorporates a control scheme (using 4-bit patterns) to load the codebook and initialize the array; this scheme, however, is more complicated that the one in our design and also does not allow the vector dimension to change.

VI. Cost and Performance Comparisons

The code rate or resolution $r = n/M$ measures the number of bits per vector element used to represent the input data. Given a fixed code rate r, the quality improves as the vector dimension M is increased [2]. In this section, we assume that the code rate is kept at $r = 0.5$ bit per sample and the word width of input data is $K = 8$ bits. In such a case, the codebook size $N = 2^n$ grows exponentially. The four implementation alternatives are different with regard to cost-effectiveness due to area and time tradeoffs. The area-time (AT) figure of merit is obtained by dividing the total circuit size by the throughput rate. This measure, therefore, estimates the normalized cost for one sample per clock cycle. Figure 6 shows the AT measure for the four designs when M varies from 16 to 36. The effect of wiring is not accounted for in deriving these figures.

The AT measure is practically significant only if the entire array can be realized on a single chip. Because current VLSI technology does not allow many millions of random logic gates on a single chip, the implementation eventually expands to multiple chips as the vector dimension is increased. Therefore, packaging becomes an important issue. The propagation time

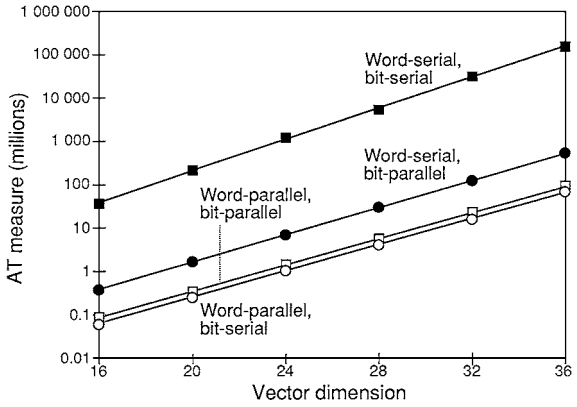

Figure 6. Various designs for VQ encoding compared with respect to the area-time (AT) figure of merit.

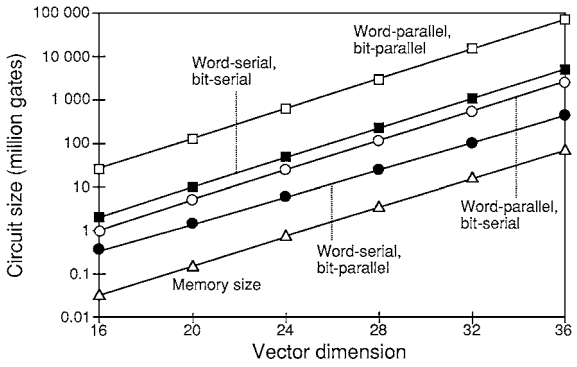

Figure 7. Circuit size (excluding memory) and memory requirement for VQ encoders of various designs.

of signals from one chip to another or from one board to a different one constitutes a significant portion of the clock cycle [23]. This may limit the maximum clock rate.

The estimated circuit size of the four designs is depicted in Fig. 7, together with the total memory size for comparison. Because the memory size is at least an order of magnitude smaller than the circuit size, it is advantageous to distribute the codebook throughout the array, thus eliminating the need for a large external memory and complicated addressing logic [11].

From Fig. 7, we note that the word-serial bit-parallel design has the lowest complexity and may in fact allow single-chip implementation if the vector dimension is not too large. The word-parallel bit-serial design, though somewhat more complex, may still be viable, given its cost/performance advantage as depicted in Fig. 6.

VII. Conclusion

We have presented a linear array structure for full-search vector quantization that is highly suitable for VLSI realization. To make the design truly scalable, control signals are not broadcast but are propagated along with the data through local and low-fanout connections. Our design encompasses a data-driven control scheme to maintain pipelined and parallel operation while providing the capability for handling the run-time loading of codevectors, array initialization, and adaptive update of the codebook and vector dimension. We have also explored the design space for the data-driven approach in terms of processor implementation alternatives.

Our data-driven control scheme allows the linear array to be controlled completely via the boundary PEs, thus making the implementation scalable with no performance penalty. Whereas pipelining is routinely used to improve performance, integrating data and control flows in a seamless manner had not been previously accomplished for VQ. This integration allows the advantages of our approach to persist, and even become amplified, with each denser generation of VLSI technology [24]. Thus, our approach does not just offer a one-time improvement but rather a series of increasing improvements as technology is continually scaled down. Performance benefits of the data-driven design method relative to a broadcast-based approach have been previously quantified [15].

Besides improved scalability and realizability (cost-effectiveness), the data-driven approach offers at least two other benefits that are worthy of further investigation. One benefit is simple conversion of the processor array to a fully asynchronous design, sometimes referred to as wavefront array [16], given the absence of broadcasting or other forms of global communication. Such a conversion will introduce some new overhead but will also eliminate the implementation cost and design difficulties associated with clock distribution in large parallel systems. It also has significant implications for power consumption. Another benefit is the potential for simple and natural integration of the design with error detection and fault diagnosis schemes [20, 25] to achieve reliable computation.

References

1. K. Dezhgosha, M.M. Jamili, and S.C. Kwatra, "A VLSI Architecture for Real-Time Image Coding Using a Vector Quantization Based Algorithms," *IEEE Trans. Signal*

Processing, vol. 40, 1992. pp. 181–189.
2. A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, New York, NY: Kluwer Academic, 1991.
3. A. Gersho and V. Cuperman, "Vector Quantization: A Pattern Matching Technique for Speech Coding," *IEEE Communications Magazine*, vol. 15, Dec. 1983, pp. 15–21.
4. R.M. Gray, "Vector Quantization," *IEEE Acoustics, Speech, and Signal Processing Magazine*, vol. 1, Apr. 1984, pp. 4–29.
5. R.M. Gray, *Source Coding Theory*, Boston, MA: Kluwer Academic, 1990.
6. N.M. Nasrabadi and R.A. King, "Image Coding Using Vector Quantization: A Review," *IEEE Trans. Communication*, vol. 36, 1988, pp. 957–971.
7. W.-C. Fang, C.-Y. Chang, B.J. Sheu, O.T.-C. Chen, and J.C. Curlander, "VLSI Systolic Binary Tree-Searched Vector Quantizer for Image Compression," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 2, 1994, pp. 33–44.
8. B. Parhami, *Introduction to Parallel Processing: Algorithms and Architectures*, New York, NY: Plenum Press, 1999.
9. C.-H. Yeh and B. Parhami, "Synthesizing High-Performance Parallel Architectures under Inter-Module Bandwidth Constraints," in *Proc. 10th Int'l Conf. Parallel and Distributed Computing and Systems*, Oct. 1998, pp. 414–416.
10. A.L. Fisher and H.T. Kung, "Synchronizing Large VLSI Processor Arrays," *IEEE Trans. Computers*, vol. 34, 1985, pp. 734–740.
11. G.A. Davidson, P.R. Cappello, and A. Gersho, "Systolic Architectures for Vector Quantization," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 36, 1988, pp. 1652–1664.
12. H. Park and V.K. Prasanna, "Modular VLSI Architectures for Real-Time Full-Search-Based Vector Quantization," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 3, 1993, pp. 309–317.
13. P.A. Ramamoorthy, B. Potu, and T. Tran, "Bit-Serial VLSI Implementation of Vector Quantizer for Real-Time Image Coding," *IEEE Trans. Circuits and Systems*, vol. 36, 1989, pp. 1281–1290.
14. M. Yan, J.V. McCanny, and U. Hu, "VLSI Architectures for Vector Quantization," *J. VLSI Signal Processing*, vol. 10, 1995, pp. 5–23.
15. D.-M. Kwai and B. Parhami, "High-Performance Array Processing with Fully Pipelined Data Streams and Control Paths," in *Proc. Int'l Conf. Parallel and Distributed Computing and Systems*, Nov. 1999, pp. 609–612.
16. S.Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice-Hall, 1988.
17. H.T. Kung and C.E. Leiserson, "Algorithms for VLSI Processor Arrays," in *Introduction to VLSI Systems*, C. Mead and L. Conway (Eds.), Reading, MA: Addison-Wesley, 1980, pp. 271–292.
18. R.B. Urquhart and D. Wood, "Systolic Matrix and Vector Multiplication Methods for Signal Processing," *IEE Proc.: Pt. F*, vol. 131, 1984, pp. 623–631.
19. R.G. Cooper, "The Distributed Pipeline," *IEEE Trans. Computers*, vol. 26, 1977, pp. 1123–1132.
20. Y.H. Choi, S.-H. Han, and M. Malek, "Fault Diagnosis of Reconfigurable Systolic Arrays," in *Proc. Int'l Conf. Computer Design*, Oct. 1984, pp. 451–455.
21. M.O. Ahmad and D.V. Poornalah, "Design of an Efficient VLSI Inner-Product Processor for Real-Time DSP Applications," *IEEE Trans. Circuits and Systems*, vol. 36, 1989, pp. 324–329.
22. B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, New York, NY: Oxford University Press, 2000.
23. H.B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Reading, MA: Addison-Wesley, 1990.
24. D.-M. Kwai and B. Parhami, "Scalability of Programmable FIR Digital Filters," *J. VLSI Signal Processing*, vol. 21, no. 1, 1999, pp. 31–35.
25. L. Li, "Systolic Computation with Fault Diagnosis," *Parallel Computing*, vol. 14, 1990, pp. 235–243.
26. M. Aloqeely and C.Y. Chen, "Sequencer-Based Data Path Synthesis of Regular Iterative Algorithms," in *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994, pp. 155–160.

Ding-Ming Kwai received the BS and MS degrees in Taiwan from the National Cheng Kung University, Tainan, and the National Chiao Tung University, Hsinchu, in 1987 and 1989, respectively, and the PhD degree from the University of California, Santa Barbara, in 1997. He was with the Chung Chen Institute of Technology, Taoyuan, Taiwan, as a reserve officer during 1989–91 and with the Hualon Microelectronics Corporation, Hsinchu, Taiwan, as a design engineer during 1991–93. He is now affiliated with the Worldwide Semiconductor Corporation, Hsinchu, Taiwan. His research interests include parallel processing, VLSI architectures, and fault-tolerant computing.

Behrooz Parhami received his PhD in computer science from University of California, Los Angeles, in 1973. Presently, he is Professor in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His research deals with parallel architectures and algorithms, computer arithmetic, and reliable computing. In his previous position with Sharif University of Technology in Tehran, Iran (1974–88), he was also involved in the areas of educational planning, curriculum development, standardization efforts, technology transfer, and various editorial responsibilities, including a five-year term as Editor of *Computer Report*,

a Farsi-Language computing periodical. His technical publications include over 200 papers in journals and international conferences, a Farsi-language textbook, and an English/Farsi glossary of computing terms. His latest publications include two textbooks on computer arithmetic (Oxford, 2000) and parallel processing (Plenum, 1999). Dr. Parhami is a Fellow of both the IEEE and the British Computer Society, a member of the Association for Computing Machinery, and a Distinguished Member of the Informatics Society of Iran for which he served as a founding member and President during 1979–84. He also served as Chairman of IEEE Iran Section (1977–86) and received the IEEE Centennial Medal in 1984.
parhami@ece.ucsb.edu