AN INTEGRATED SYSTOLIC ARRAY DESIGN FOR VIDEO COMPRESSION

Pol Lin Tai, Chii Tung Liu, Shih Yu Huang, and Jia Shung Wang

Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan 30043, R.O.C.

ABSTRACT

This paper presents an integrated systolic array design for implementing full-search block matching, 2-D discrete wavelet transform, and full-search vector quantization on the same VLSI architecture. The above three functions are prime essential but with large amount of computation in video compression. To meet real-time requirement, many systolic arrays are designed for individually performing each of them. In fact, these functions contain the similar computational procedure. The matrix-vector product forms of them are quite similar. In this paper, with carefully extracting the common computation component, an integrated systolic array design that can perform above three functions is presented. A utilization of 100% to 97% is achieved for executing full-search block matching and full-search vector quantization. When performing 2-D discrete wavelet transform, the utilization is about 32%. The proposed integrated architecture spends lower hardware cost and has a regular hardware structure. It befits the VLSI implementation for video compression.

1. INTRODUCTION

In video compression, some kernel functions such as, block matching, discrete wavelet transform, vector quantization, etc. are prime essential but with large amount of computation. For real-time applications, custom hardware devices may be required to implement these functions. In recent years, many VLSI architectures for performing full-search block matching (FSBM), discrete wavelet transform (DWT) and full-search vector quantization (FSVQ) have been proposed. All the past proposed were designed for individually executing one of the above three functions. For instance, the architectures in [1][2] are designed for FSBM; [3][4] are proposed for executing DWT, [5][6] are presented for FSVQ. In other word, if we wish to implement wavelet-based codec or vector quantization based codec, several VLSI architectures should be enclosed.

To reduce the hardware cost, the best way for executing the above three functions might be designing an integrated hardware architecture that could perform all the three functions. In fact, if we dissect these functions into the basic matrix-vector product forms, it could be easily found that the three functions contain the similar computational procedure. All of them could be viewed as one kind of FIR filter. In other word, an integrated hardware design for them becomes practicable. In this paper, an integrated systolic array design that could execute FSBM, 2-D DWT and FSVQ in the same hardware architecture is presented. The common computational components are extracted carefully to reduce the complexity of the architecture. The computational schedules are designed discreetly to enhance the system throughput. The input signals are serial-in when executing FSBM. While performing 2-D DWT and FSVQ, the system is divided into four similar modules and the signal-input mode is transferred to parallel mode. Since the unified architecture spends lower hardware cost and has a regular hardware structure, it is suited for VLSI implementation for video/image compression applications that require all the functions.

2. Computational Procedure Analysis

In this section, the DWT, FSBM, and FSVQ are briefly described firstly. Then the computational procedures of the three functions are analyzed from the viewpoint of matrix-vector product form.

2.1. Discrete Wavelet Transform

The 1-D DWT transforms input signal into average and detail component by applying original signal with low-pass and high-pass filter. The average component then applies DWT again to obtain next level decomposition. It can be expressed as follows:

$$W_L(j,k) = \sum_{n=0}^{p-1} h(n) W_L(2j+n, k-1) \qquad (1)$$

where $W_L(j,k)$ is the jth low-pass coefficient of the kth decomposition level, p is the filter size. The 2-D DWT is calculated by extending the 1-D DWT directly. For instance, the LH part can be expressed as:

$$W_{LH}(k,i,j) = \sum_{m=0}^{p-1}\sum_{n=0}^{p-1} h(m) g(n) W_{LH}(k-1, 2i+m, 2j+n) \qquad (2)$$

For simplicity, the operation time of one multiplication and one accumulation is set as one clock. The total cycles for executing 2-level 2-D DWT with frame size $W\times H$ and filter size P is

$$W\times H\times(p^2+1/4\times p^2) = W\times H\times 5/4\times p^2. \qquad (3)$$

2.2. Full-Search Block Matching

FSBM divided current frame into non-overlapped blocks of size $N\times N$. Each block exhaustively searches its own displacement vector within an area of size $(2P+N-1)\times(2P+N-1)$ in the referenced frame. The MAE for displacement (u,v) is denoted as:

$$MAE(u,v) = \frac{1}{N\times N}\sum_{i=1}^{N}\sum_{j=1}^{N}\left|a(i,j) - b(i+u+P, j+v+P)\right| \qquad (4)$$

The motion vector is defined as:

$$mv(x,y) = \arg\min_{(u,v)} MAE(u,v), \text{ and } -P \le u, v \le P-1. \qquad (5)$$

Assume the operations of one subtraction, one magnitude operation, and one accumulation can be completed in one clock. The total cycles for executing FSBM with frame size $W\times H$ is

$4\times W\times H\times P^2$. (6)

2.3. Full-Search Vector Quantization

A vector quantization (VQ) is a mapping from the k-dimension Euclidean space to a finite subset *S*. Let $S=\{S_i|i=1,2,\ldots,N_s\}$ denotes the codebook where N_s is the size of the codebook. $S_i=(S_i^1 S_i^2 \ldots S_i^k)$ indicates the codeword in the codebook. For each input vector $X=(x^1, x^2,\ldots,x^k)$, the FSVQ exhaustively searches the closest codeword in the whole codebook to represent the input vector. The distortion between *X* and S_i is defined as:

$$d_i = d(X,S_i) = \Sigma|(X_j - S_i^j)| \quad (7)$$

The operations of one subtraction, one magnitude operation, and one accumulation are assumed can be completed in one clock. The total cycles for executing FSVQ with frame size $W\times H$ is

$$W\times H/K\times K\times N_S = W\times H\times N_S. \quad (8)$$

2.4. Common Computational Components

Firstly, we transfer (1) into the matrix-vector product form. Then, a 4-tap DWT can be expressed as:

$$\begin{bmatrix} W_L(0,j) \\ W_L(1,j) \\ W_L(2,j) \\ \vdots \end{bmatrix} = \begin{bmatrix} h(0) & h(1) & h(2) & h(3) & & & \\ & & h(0) & h(1) & h(2) & h(3) & \\ & & & & h(0) & h(1) & h(2) & h(3) \\ & & & & & & \ddots \\ h(2) & h(3) & & & & & h(0) & h(1) \end{bmatrix} \times \begin{bmatrix} W_L(0,j-1) \\ W_L(1,j-1) \\ W_L(2,j-1) \\ \vdots \end{bmatrix} \quad (9)$$

For transfering FSBM into matrix-vector pjroduct form, let the absolute difference operator as $\otimes$, $\beta_i=b(i,j+v+p)$, $\alpha_i=a(i,j)$. Then (4) could be simplified to of the following form:

$$C_k = \sum_{i=1}^{N} a_i \otimes b_{i+k-1}, \ \forall\, k=1, \ldots, 2P. \quad (10)$$

From the viewpoint of matrix-product form, it becomes,

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & \cdots & a_N & & \\ & a_1 & a_2 & a_3 & a_4 & \cdots & a_N & \\ & & a_1 & a_2 & a_3 & a_4 & \cdots & a_N \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \end{bmatrix}. \quad (11)$$

The matrix-vector product form of FSVQ can be expressed as

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} x1 & x2 & \cdots & xk & & & \\ & x1 & x2 & \cdots & xk & & \\ & & & x1 & x2 & \cdots & xk \\ & & & & & & \ddots \end{bmatrix} \otimes \begin{bmatrix} M(S_1) \\ M(S_2) \\ M(S_3) \\ \vdots \end{bmatrix} \quad (12)$$

where $M(S_i)$ means $[S_i^1 S_i^2 \cdots S_i^k]^T$ and $\otimes$ denotes as MAE.

It is easily seen that there is strong correlation among FSBM, DWT, FSVQ, and FIR filter. It suggested that one could design a VLSI architecture that is able to perform the functions of FSBM, DWT, and FSVQ based on the structure for executing FIR filter. Based on such observation, an integrated systolic array is presented for executing FSBM, 2-D DWT and FSVQ.

3. The Integrated Systolic Array Design

Fig. 1 shows the block diagram of the proposed VLSI architecture. It consists of 16 processing elements, 32 cyclic shift registers, 4 delay registers and 4 minimum distortion detectors. The proposed architecture can execute the following functions. (1) the FSBM with current block size of 16×16 and search range (-8,7). (2) the 2-D 2-level Harr transform with block size 8×8. (3) the FSVQ with input vector size 2×2.

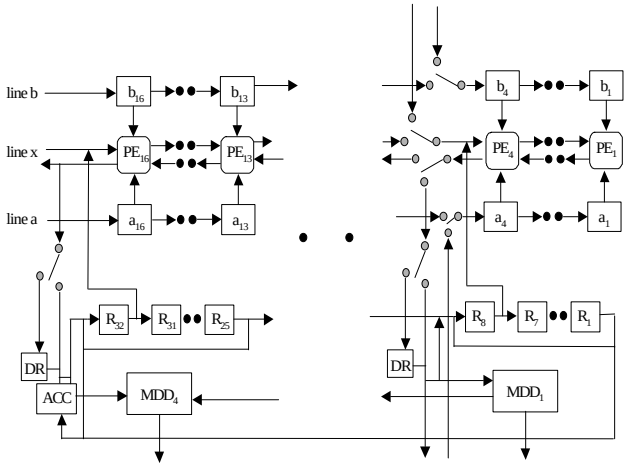


Fig. 1. Block diagram of the proposed VLSI architecture.

Fig. 2 shows the block diagram of one processing element (PE). It consists of two registers, one adder, and one product operator. The type of product operator depends on the executing function. The function of PE is to calculate the product for the values stored in the two registers and then add the result to the partial sum. The partial result will be sent to the PE at its left-hand side. The function of the cyclic shift register (CSR) is to store the intermediate results generated from the leftmost PE. For FSBM, the intermediate results indicate the partial sum for one referenced block. When performing 2-D DWT, the coefficients of the first level LL image will flow into CSR for the operation of the second level DWT. When performing the FSBM, the MDD_1 stores the minimum MAE value. When executing FSVQ, the four MDDs store the four minimum MAE values for the input vector.

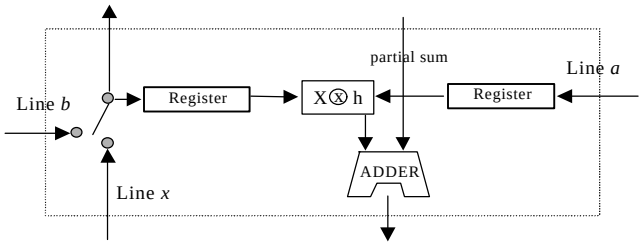


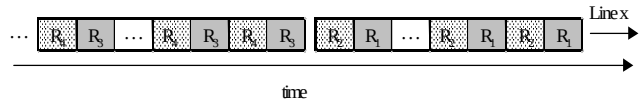
Fig. 2. The block diagram of the processing element.

4. Computational Schedule Design

4.1. Computation Schedule for FSBM

The basic input unit for FSBM is one row of the search area. The search area data are fed into PEs by the way shown in Fig. 3 (a). Two rows of the search area are fed into PEs interlaced. The utilization of PEs is demonstrated in Fig. 3 (b). When odd PEs calculate the partial sums of the first row data, the even PEs calculate the partial sums for the second row. In the next time, the odd PEs will turn to calculate the partial sums of the second row data. By the meanwhile, the partial sums of the first row are

calculated by even PEs. By such approach, all PEs are fully utilized and PE_{16} will finish one partial sum every clock cycle.



(a)

	PE_{16}	PE_{15}	PE_{14}	PE_{13}	PE_{12}	PE_{11}	PE_{10}	PE_9	PE_8	PE_7	PE_6	PE_5	PE_4	PE_3	PE_2	PE_1
31	(1,1)	(1,2)	(2,1)	(2,2)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)
32	(1,2)	(2,1)	(2,2)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)	(9,1)
33	(2,1)	(2,2)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)	(9,1)	(9,2)
34	(2,2)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)	(9,1)	(9,2)	(10,1)
35	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)	(7,1)	(7,2)	(8,1)	(8,2)	(9,1)	(9,2)	(10,1)	(10,2)

(i,j): PS^1(i,j)

(b)

Fig. 3. The computational schedule for FSBM. (a) the input signal scan path for the search area data (b) an example of the utilization of PEs from time 31 to time 35.

Let $S(i,j,k)$ denote the sum of the first k partial sums of the referenced block (i,j). At cycle time 1×32+31, the partial sum $PS^2(-8,-8)$ is completed by PE_{16}. This value will be sent into the accumulator in the next cycle for the calculation of S(1,1,2). It can be accomplished because $PS^1(-8,-8)$ will reach register R_{32} in that time. Repeating the above procedure, at cycle time 15×32+31, PS^{16} $(-8,-8)$ is calculated completely and will be summed with S(1,1,15) at next cycle. That is, the MAE for the top-left referenced block, MAE(-8,-8), is accomplished at cycle time 16×32. Since the PEs are fully utilized, one MAE will be generated in the accumulator every clock cycle from now on. At cycle time 16×32×8+31, the mv will be generated in MDD_4.

The total time of completing FSBM for the first current block is $N\times4P\times N/2+31$. For other blocks, they do not take up addition initial clocks. Therefore, the total cycles for executing FSBM with frame size $W\times H$ is

$$Time_{FSBM}=N\times4P\times N/2\times W\times H/N/N+31 = 2\times P\times W\times H+31. \quad (13)$$

The utilization of the proposed system to perform FSBM is $100\times(4\times W\times H\times8^2)/((2\times8\times W\times H+31)\times16)\% \cong 100\%$.

4.2. Computation Schedule for 2-D DWT

As demonstrated in Fig. 5 (a), the proposed system is divided into four processing modules (PM) when executing 2-D DWT. Each PM contains four PEs, four registers of group *b*, four registers of group *a*, and eight CSRs. Input signal will be parallel fed into four PMs as indicated in Fig. 5 (b). Each PM performs 2-D DWT directly by (2). Following we focus the discussion on the computational schedule of the first PM.

Fig. 5 (b) shows the input signal structure for the first level DWT. Each input unit contains four pixels. Fig. 4 shows the computation schedule of the first level DWT. The input signal module shown in Fig. 5 (b) is fed into PEs four times. The four filter banks will load into PEs from line *a* concurrently with the input signal module. The initial time for HH image can be removed for HL, LH, and LL by overlapping the input signal module. Therefore, the HH, HL, LH, LL are generated from cycle time 7 to 12, 15 to 20, 23 to 28, 31 to 36, respectively.

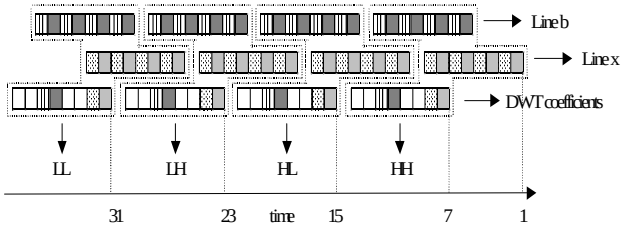
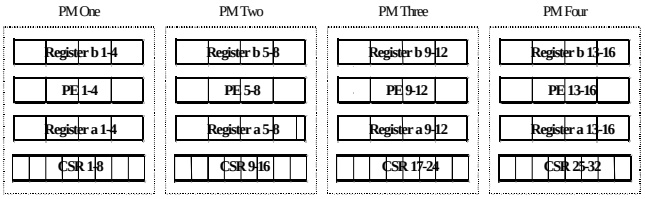
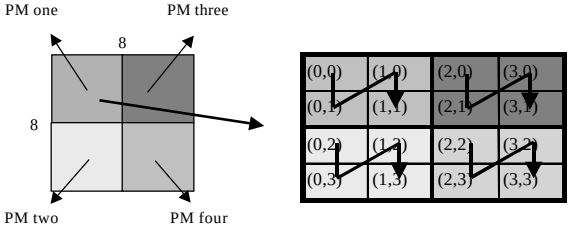


Fig. 4. The computational schedule for the first level DWT.

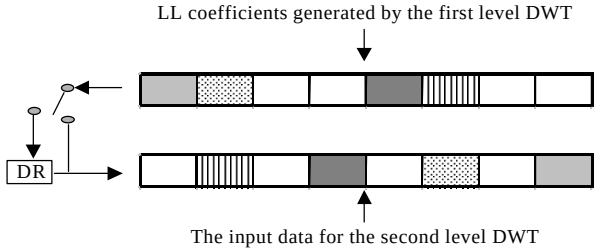


(a)

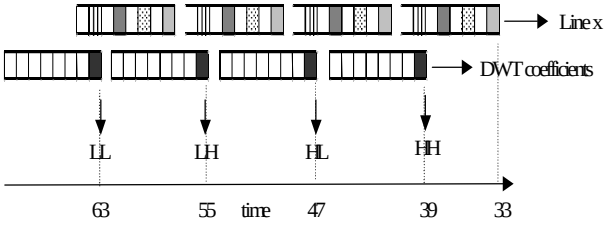


(b)

Fig. 5. The PMs and the input signal partition for DWT. (a) the structure of the PMs (b) the signal partition of one 8×8 block.



(a)



(b)

Fig. 6. The computational schedule for the second level 2-D Harr transform. (a) The function illustration of the delay register. (b) the computational schedule of the second level DWT.

The coefficients of LL image will flow into CSRs and feed into PEs as input signal for the second level DWT. In order to execute 2-D DWT directly, the coefficient sequence of LL generated from

time 31 to 36 is re-arranged by the delay register to satisfy the data flow requirement of the second level DWT. The re-arranged coefficient sequence is shown in Fig. 6 (a). The coefficients of the first level LL image stored in CSR are fed into PEs every 8 cycle times. The computational schedule for level two DWT is shown in Fig. 6 (b).The HH, HL, LH, LL part of the second level DWT can be generated at time 39, 47, 55, 63, respectively.

Since the input signal for each PM are independent, the four PMs could compute its own 2-D DWT coefficients concurrently. Therefore, the total time for one frame of size $W\times H$ to execute 2-level Harr transform is

$$Time_{DWT} = W\times H/64\times 63. \quad (14)$$

The utilization of the proposed system to perform 2-level DWT is $100\times(W\times H\times 5/4\times 2^2)/((\ W\times H/64\times 63)\times 16)\% \cong 32\%$.

C. Computational Schedule for FSVQ

For the convenience of the following discussion, the codebook size is assumed as 1024. When executing FSVQ, the system is divided into four processing modules (PM) as shown in Fig. 7. Each PM contains four PEs, four registers of group *a*, four registers of group *b*, and one MDD. The codewords with index $4k+i$, for k=0 to 255, are searched by PM i to find the closest codeword. Following we focus the discussion on the computational schedule of the PM one.

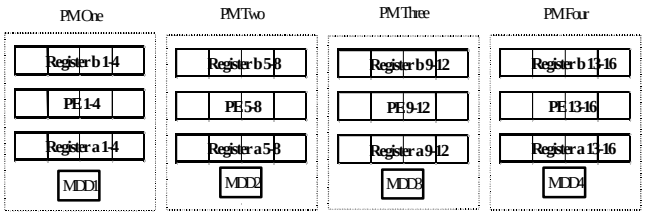
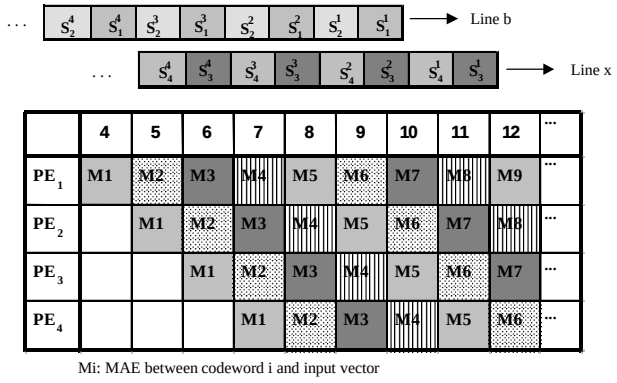


Fig. 7. The processing modules for performing FSVQ.



Mi: MAE between codeword i and input vector

Fig. 8. The computational schedule for executing FSVQ.

The basic input unit for performing FSVQ is one codeword. The input schedule of codewords is demonstrated in Fig. 8. Four codewords are composed as one input module. By such input scan path, the four PEs can calculate four MAEs of four different codewords concurrently. After cycle time 8, PE_4 will complete one MAE every cycle. When the last MAE is accomplished at cycle time 263, the local closest codeword can be decided in MDD_1 at cycle time 264.

The four PMs are executing FSVQ concurrently. At cycle time 264, four codewords, which are the local closest codewords among four 256 codewords input sets, are decided in four MDD. The minimum values stored in MDD_3, MDD_2, MDD_1 are fed into MDD_4 sequentially to decide the global closest codeword. At cycle time 267, the closest codeword for the first input vector is decided in MDD_4. The initial time for the other input vectors is reduced to 5. Therefore, the total cycle time for one frame of size $W\times H$ to execute FSVQ with codebook size 1024 is

$$Time_{FSVQ} = W\times H/4\times 265+2. \quad (15)$$

The utilization of the proposed system to perform FSVQ is $100\times(W\times H\times 1024)/((W\times H/4\times 265+2)\times 16)\% \cong 97\%$.

5. CONCLUSIONS

In this paper, an integrated systolic array for performing FSBM, 2-D DWT, FSVQ in the same architecture has been presented. We first transfer these functions into matrix-vector product form and show that there are similar computation procedures among these functions. By such mapping technique, the common components of the three functions are then carefully extracted. In order to enhance the throughput of the integrated system, two kind of computational schedules are designed. When executing FSBM, the input signals are serial-in to the processing elements. When performing 2-D DWT and FSVQ, the system is divided into four processing modules. The input signals are parallel fed into four processing modules. Each module performs its own computation individually. In our system, a utilization of almost 100% can be achieved for FSBM and FSVQ. When performing 2-D DWT, the system utilization is about 32%. The architecture is simple, modular, and low cost. It is very suit the real-time applications where low cost hardware video codec is required.

6. REFERENCES

[1] Seung Hyun Nam, Jong Seob Baek, Moon Key Lee, "Flexible VLSI architecture of full search motion estimation for video applications," *IEEE Trans. on Consumer Electronics*, vol. 40, no. 2, pp. 176-183, May 1994.

[2] Viet L. Do and Kenneth Y. Yun, "A low-power VLSI architecture for full-search block-matching motion estimation," *IEEE. Trans. on Circuits Syst. Video Technol.*, Vol. 8, no. 4, pp. 393-398, Aug. 1998.

[3] Aleksander Grzeszczak, Mrinal K. Mandal, Sethuranman Panchanathan, and Tet Yeap, " VLSI implementation of discrete wavelet tansform," *IEEE Trans. on VLSI Syst.*, vol. 4, no. 4, pp. 421-433, DEC. 1996.

[4] Gauthier Lafruit, Francky Catthoor, Jan P. H. Cornelis, and Hugo J. De Man, "An efficient VLSI architecture for 2-D wavelet image coding with novel image scan," *IEEE Trans. on VLSI Syst.*, vol. 7, no. 1, pp. 56-68, March 1999.

[5] Chin-Liang Wang, and Ker-Min Chen, "A New VLSI Architecture for Full-Search Vector Quantization," *IEEE. Trans. on Circuits Syst. Video Technol.*, vol. 6, no. 4, pp. 389-398, Aug. 1996.

[6] Wai-Chi Fang, Chi-Yung Chang, Bing J. Sheu, Oscal T.-C. Chen, and John C. Curlander, "VLSI Systolic Binary Tree-Searched Vector Quantizer for Image Compression," *IEEE Trans. on VLSI Syst.*, vol. 2, no. 1, pp. 33-44, March 1994.