algorithm step by step, we have

$$k_0^0 \leftarrow \frac{1}{\sigma_0}$$
$$f_1 \leftarrow [\rho]$$
$$\beta_1 \leftarrow \sigma_0 - \rho\sigma_1$$
$$k_0^1 \leftarrow \frac{1}{\sigma_0} + \frac{\rho^2}{\sigma_0 - \rho\sigma_1}$$
$$b_2 \leftarrow [0 \quad \rho]^T$$
$$\alpha_1 \leftarrow \sigma_0 - \rho\sigma_1$$
$$k_0^1 \leftarrow \frac{1+\rho^2}{1-\rho^2}\frac{1}{\sigma_0}$$
$$k_1^1 \leftarrow -\frac{\rho}{\sigma_0 - \rho\sigma_1}$$

and

$$\varsigma_{e,1} = \frac{1-\rho^2}{1+\rho^2}\sigma_0 \qquad c_1^1 = \frac{\rho}{1+\rho^2}$$

as expected.

III. Conclusion

We demonstrated how the Sherman–Morrison identity for inversion of the symmetric matrices can be utilized to develop a new fast algorithm for optimal linear interpolation. We demonstrated that the proposed algorithm has lower complexity with respect to the other proposed schemes such as [4]. It is also possible to develop lattice type structure for linear interpolation based on the proposed algorithm.

References

[1] G. A. Merchant and T. W. Parks, "Efficient solution of a Toeplitz-plus-Hankel coefficient matrix system of equations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 40–44, Feb. 1982.
[2] H. Akaike, "Block Toeplitz matrix inversion," *SIAM J. Appl. Math.*, vol. 24, pp. 234–241, Mar. 1973.
[3] A. E. Yagle, "New analogs of split algorithms for arbitrary Toeplitz-plus-Hankel matrices," *IEEE Trans. Signal Processing*, vol. 39, pp. 2457–2463, Nov. 1991.
[4] S. L. Marple, "Fast algorithms for linear prediction and system identification filters with linear phase," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 942–953, Dec. 1982.
[5] C. K. Coursey and J. A. Stuller, "Linear interpolation lattice," *IEEE Trans. Signal Processing*, vol. 39, pp. 965–967, Apr. 1991.
[6] P. Strobach, *Linear Prediction Theory, A Mathematical Basis for Adaptive Systems.* Springer-Verlag, 1990.

Binary Partition Algorithms and VLSI Architectures for Median and Rank Order Filtering

Charng Long Lee and Chein-Wei Jen

Abstract—**A class of selection algorithms by binary partition is very efficient for median and rank order filtering. A unified discussion of these algorithms is presented. Binary partition algorithms have better time-area complexity than sorting methods. Counting, firing, and updating are three basic steps. A generic structure is proposed to realize these algorithms. They can be implemented by simple and regular modules in VLSI.**

Manuscript received November 23, 1991; revised November 6, 1992. The associate editor coordinating the review of this correspondence and approving it for publication was Prof. Gonzalo Arce. This work was supported by the National Science Council, Taiwan, Republic of China, under Grant NSC 79-0404-E009-27.

C. L. Lee was with the Institute of Electronics Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, Republic of China. He is now with the Video Signal Processing Department, CCL/ITRI, Taiwan, Republic of China.

C.-W. Jen is with the Institute of Electronics and Department of Electronic Engineering, National Chiao Tung University, Hsinehu, Taiwan, Republic of China.

IEEE Log Number 9210117

I. Introduction

Rank order filtering is a nonlinear filtering technique which picks up an output according to the order statistics of elements in a sliding window. Maximum, minimum,, and median filters are some frequently used examples. Median filters are robust, and can remove impulse noise while preserving some essential features [1]. They are widely applied in many signal and image processing applications, such as seismic data [2], medical images [3], or video signals [4]. Maximum and minimum selections can be found in gray-scale morphological dilation and erosion [5]. Since the rank order and median filtering processes have played important roles in many signal and image applications, fast realizations of them are necessary.

The implementation of median or rank order filters can be divided into two categories. The first category realizes them in software on general-purposes sequential or parallel computers. The second one implements the filters on VLSI hardware. In the software category, the basic procedure for order statistic calculation is comparison and sorting. However, if only a specific rank order is required, selection algorithm will be more effective. This is because sorting can be constructed by selection [6], [7] and the selection algorithms are linear in complexity [8]. In addition to these computer science methods, there are some fast algorithms which take advantage of the running window where only a minor portion of the elements are deleted and replaced by the same number of new elements. An updated histogram method in [9] and a moving border method in [10] were based on this concept for two-dimensional median filtering. Some heap-based methods have been proposed in [11] and [12]. Parallel implementation of median filters can also be found in [13] and [14].

The hardware category has several approaches. The first one is to implement order statistic filtering by systolic algorithms with a linear array of identical processing elements [15]. The next approach is to realize median filtering by a regular sorting network, such as bubble sort or odd-even transposition sort [16]–[18]. The third approach is a class of "radix methods" which calculate the result based on the binary representation of data. Many algorithms [19]–[22] and VLSI architectures [23]–[26] belong to this approach. An interesting method takes binary median on a stack of threshold decomposed signals and then sums up the result at each threshold level as the final output [27]. This method is not practical because of its exponential hardware complexity. An *m*-array method has also been implemented on VLSI circuits [28]. It performs binary search on an accumulated histogram.

In this correspondence, we propose a unified representation for the "radix method." All these methods stem from a binary partition algorithm which is discussed in Section II. The unified description and comparison of these algorithms is presented in Section III. A generic architecture is presented in Section IV with considerations of tradeoffs. Also in Section IV, word-parallel bit-pipeline VLSI architecture designs are discussed for high-speed signal processing. Applications and extensions of the binary partition algorithms are discussed in Section V.

II. Order Statistic Calculation by Partition

Selection is a conventional problem in computer algorithms. It is to find the kth largest element in a set of elements, $S = \{x_1, \cdots, x_N\}$. A brute-force approach is to sort the elements in S to a decreasing order and then pick up the value of the kth position. However, there is a more efficient way which uses the idea of divide and conquer [8]. We call it the partition-selection approach.

A. Arbitrary Partition

The idea of partition is to divide the original set S into two subsets, S_0 and S_1, by arbitrarily choosing an element x_b in S as the boundary. Those elements which are equal to or greater than x_b are put in S_1, otherwise they are put in S_0. A conceptual diagram is shown in Fig. 1. If the number of elements in S_1, denoted as $|S_1| = q$, is greater than k, that is $q > k$, then the original problem size will be reduced to a smaller one which is to select the kth largest element in the subset S_1. If $q < k$, then the new problem is to select the $(k - q)$th largest element in the subset S_0. Repeat the partition process until $q = k$, that means the desired output x_d is equal to the boundary element x_b. In the worst case, it will not stop until $q = k = 1$. This is arbitrary partition for selection.

B. Binary Partition

The partition of set S can be more than two. However, our interest is in the two-subset case. We assume the data to be nonnegative integers because most of the gray-scale images and signals are positive. They can be either normalized or shifted to be nonnegative without loss their generality. Suppose the word length of binary representation is L, then the value of data can be no more than $2^L - 1$. A special case of two-subset partition chooses the boundary element to be the power of 2 ($2^j, j = 1 \cdots L - 1$), or their linear combinations. We consider this as binary partition.

Arbitrary and binary partition are similar. The major difference is that x_b of the former method is one of the element in set S, but x_b of the latter one is determined externally. The number of iterations for arbitrary partition is not fixed. However binary partition approach needs L iterations at most to get an output. A conceptual description shown in Fig. 2 demonstrates the idea of binary partition. Because we are selecting the kth largest element in S, the data profile is monotonously decreasing. The finding of desired output is equivalent to performing a binary search along the vertical axis and check of matching along the horizontal axis.

III. Binary Partition Algorithms for Hardware Implementations

Basically this method obtains the result with one bit at a time. It needs a mask vector to define the candidate subset where possible targets are lying in. That is either S_1 or S_0. The candidate subset will shrink as the investigation proceeds from MSB's to LSB's. At last, the only element left is the answer. If more than one element is left, their magnitudes must be the same. Many of "radix methods" or bit-level algorithms are based on this idea [19]-[26]. The major difference is the counting schemes they perform. We will unify these algorithms after we define some notations.

A. Binary Representation

The set of elements in a sliding window is defined as $W = \{x_1, \cdots, x_i, \cdots, x_N\}$, where N is the size of the window. The binary representation of an element x_i is as

$$x_i = \sum_{j=1}^{L} a_{i,j} 2^{L-j}, \qquad a_{i,j} \in \{0, 1\}$$

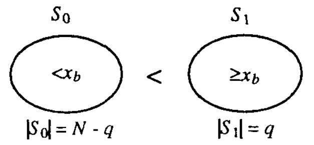

Fig. 1. Partition the set S into two subset S_0 and S_1. Number of elements in S_1 is q, (denoted as $|S_1| = q$). The magnitude of elements in S_1 are greater than those in S_0.

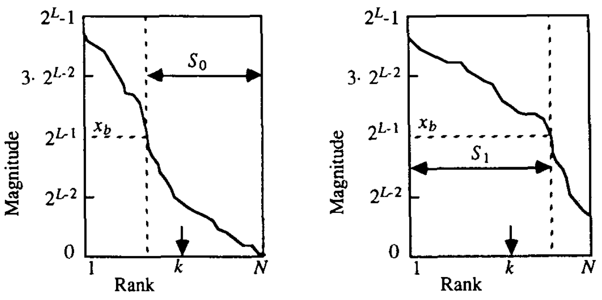

Fig. 2. Two possible sorted data profiles of set S to demonstrate the concept of binary partition. The choosing of new partition boundary depends on the subset where the desired kth largest element stays.

where L is the word length, $a_{i,1}$ is the MSB, and $a_{i,L}$ is the LSB of x_i.

Now we collect jth bits of all elements to form a column vector $\boldsymbol{a}_j = [a_{1,j} \cdots a_{N,j}]^T$. Two additional column vectors $\boldsymbol{m}_j = [m_{1,j} \cdots m_{N,j}]^T$ and $\boldsymbol{s}_j = [s_{1,j} \cdots s_{N,j}]^T$ should be defined corresponding to $\boldsymbol{a}_j$. Mask vector $\boldsymbol{m}_j$ specifies the candidate subset during the selection process. Setting vector $\boldsymbol{s}_j$ defines a set of setting values. Both $m_{i,j}$ and $s_{i,j} \in \{0, 1\}$.

B. Algorithms

Generally speaking, the binary partition algorithms generate an output value from MSB to LSB by inspecting the data in the candidate set from MSB column $\boldsymbol{a}_1$ to LSB column $\boldsymbol{a}_L$, sequentially. Counting the number of bit "ONE" or "ZERO" indicated by the mask vector in the jth bit-column can determine the jth output bit of result. From a common point of view, this kind of algorithms consist of three major parts. i) *Scoring* is to count the number of bit "ONE" or "ZERO" in the candidate subset of $\boldsymbol{a}_j$, and denote it as z_j. ii) *Firing* is to compare the counting value z_j with a threshold value k_j to determine the current binary output. iii) *Updating* is to shrink the candidate subset and determine the new threshold value for the next inspection. The candidate set is reduced through the modification of mask vectors. Repeating these steps L times will get a complete output.

1) Counting "ONE" scheme: The method reported in [19] is probably the first algorithm that selects a median from the radix-2 representation of data. This algorithm wants to find the kth largest element in the window. Thus the algorithm counts bit "ONE" in each inspection. The following steps are repeated from MSB to LSB.

i) The scoring value z_j is the number of bits which are "ONE" in the jth bit position within the candidate subset. It can be calculated as the inner product of mask vector $\boldsymbol{m}_j$ and column vector $\boldsymbol{a}_j$:

$$z_j = \boldsymbol{m}_j^T \boldsymbol{a}_j. \tag{1}$$

ii) Fire the output bit by comparing z_j with the rank value k_j:

$$f_j(z_j, k_j) = \begin{cases} 1 & \text{if } z_j \geq k_j \\ 0 & \text{if } z_j < k_j. \end{cases}$$

This expression can be rewritten as

$$f_j(z_j, k_j) = \operatorname{sgn}(z_j - k_j) \quad (2)$$

where $\operatorname{sgn}(x) = 1$ if $x \geq 0$, otherwise $\operatorname{sgn}(x) = 0$.

iii) Updating of the threshold value and mask vector for the next iteration are as follows:

If $f_j = 1$, then $k_{j+1} = k_j$ and $\boldsymbol{m}_{j+1} = \operatorname{diag}[\boldsymbol{m}_j \boldsymbol{a}_j^T]$.

If $f_j = 0$, then $k_{j+1} = k_j - z_j$ and

$$\boldsymbol{m}_{j+1} = \operatorname{diag}[\boldsymbol{m}_j(\boldsymbol{e} - \boldsymbol{a}_j)^T].$$

A column vector with all its N entries equal to "ONE" is $\boldsymbol{e} = [1 \ 1 \ \cdots \ 1]^T$ and the diagonal elements of matrix $\boldsymbol{D}$ form a column vector by diag $[\boldsymbol{D}]$. The updating equations can be revised and merged into

$$k_{j+1} = k_j + (f_j - 1)z_j \quad \text{and} \quad (3)$$

$$\boldsymbol{m}_{j+1} = \operatorname{diag}[\boldsymbol{m}_j((1 - f_j)\boldsymbol{e} - (-1)^{f_j}\boldsymbol{a}_j)^T]. \quad (4)$$

The new rank value, k_{j+1}, of the desired output may be decreased because of (3).

2) Counting "ZERO" scheme: This scheme has been reported in two places, [21] and [22], with different style of descriptions. This algorithm finds the kth smallest element in the window. So they counted bit "ZERO" instead of bit "ONE". The formulation is as follows:

i) The scoring value is the number of bit "ZERO" in column vector $\boldsymbol{a}_j$. Thus z_j is calculated as

$$z_j = \boldsymbol{m}_j^T(\boldsymbol{e} - \boldsymbol{a}_j). \quad (5)$$

This can be interpreted as the total number of nonzero elements in the mask vector minus the number of nonzero elements in the data vector which are flagged by the mask vector.

ii) Firing is done by

$$f_j = 1 - \operatorname{sgn}(z_j - k_j) \quad \text{or} \; f_j(z_j, k_j) = \begin{cases} 1 & \text{if } z_j < k_j \\ 0 & \text{if } z_j \geq k_j. \end{cases} \quad (6)$$

iii) Updating of threshold value and mask vector are derived in the same way as in the counting "ONE" scheme. Modification of rank value can be expressed as

$$k_{j+1} = k_j - f_j z_j. \quad (7)$$

The updating of $\boldsymbol{m}_{j+1}$ is the same as (4).

3) Counting both "ZERO" and "ONE" scheme: The main idea of this algorithm [26] is to keep the threshold value k constant, and calculate the scoring value until both values are the same at last. The algorithm is as follows:

i) Scoring is as

$$z_j = z_{j-1} + \boldsymbol{m}_j^T(f_{j-1}\boldsymbol{e} - \boldsymbol{a}_j). \quad (8)$$

This expression states that if the previous firing value f_{j-1} is "ONE," then it accumulates the number of bit "ZERO" in the candidate subset for the present scoring process, otherwise it subtracts the number of bit "ONE" in the subset from the previous scoring value.

ii) The firing is as simple as

$$f_j = 1 - \operatorname{sgn}(z_j - k) \quad \text{or} \; f_j(z_j, k) = \begin{cases} 1 & \text{if } z_j < k \\ 0 & \text{if } z_j \geq k. \end{cases}$$

This algorithm also searches for the kth smallest element in the window.

iii) The updating of the next mask vector $\boldsymbol{m}_{j+1}$ is the same as (4).

4) Mask-and-set scheme: The mask-and-set concept was presented in [26]. It is different from the previous schemes in several aspects. The desired rank order is fixed at the design stage. No counting is performed. The function of updating threshold is replaced by modifying setting vectors $\boldsymbol{s}_j$. Firing is done through a threshold gate [31]. A similar method has been proposed in [25]. The procedures are as follows.

i) A composite vector $\boldsymbol{d}_j = [d_{1,j} \ d_{2,j} \ \cdots \ d_{N,j}]^T$ is formed as

$$\boldsymbol{d}_j = \operatorname{diag}[\boldsymbol{m}_j \boldsymbol{a}_j^T] + \operatorname{diag}[(\boldsymbol{e} - \boldsymbol{m}_j)\boldsymbol{s}_j^T].$$

This operation is equivalent to

$$d_{i,j} = \left\{ \begin{array}{ll} a_{i,j} & \text{if } m_{i,j} = 1 \\ s_{i,j} & \text{if } m_{i,j} = 0 \end{array} \right\} \quad \text{for } i = 1 \text{ to } N. \quad (9)$$

ii) Firing is by

$$f_j = T_k(d_{i,j}, i = 1 \cdots N) \quad (10)$$

where T_k stands for a threshold logic function which will be true if the number of bit "ONE" in the vector $\boldsymbol{d}_j$ exceeds rank order k.

iii) Modification of the next mask vector $\boldsymbol{m}_{j+1}$ is the same as in the previous three schemes and the next setting vector is updated as $\boldsymbol{s}_{j+1} = \boldsymbol{d}_j$.

A special feature of this method is that threshold rank is programmed and fixed in the threshold logic function. Although the candidate set is shrunk as usual, the scope of consideration remains to be the entire entries of vector $\boldsymbol{d}_j$. The advantage of this algorithm is the simplicity of threshold logic functions rather than arithmetic operations. This will benefit hardware implementations.

IV. A Generic Architecture for Hardware Implementation

A. The Generic Architecture for Binary Partition Algorithms

The binary partition algorithms presented in the previous section consisted of three major steps. They are also the building blocks for a generic architecture as shown in Fig. 3. A scoring unit calculates scoring output by inspecting input data with reference to the mask-in and control-in signals. A firing unit generates resulting bit by comparing the desired rank value with the scoring one. The new rank value is also obtained. An updating unit modifies the mask-out and control-out signals for the next stage depending on the current output result.

For the scoring function in counting "ONE" scheme, the inner product of (1) can be easily accomplished by performing entry-wise logic AND function on vectors $\boldsymbol{m}_j$ and $\boldsymbol{a}_j$ followed by a bit-sum counting. The hardware requirements are N AND gates and a parallel counter. Similar scoring function in (5) and (8) can be accomplished in the same way. The only difference is that the input data must be inverted for counting "ZERO."

The firing block can be easily realized by a subtracter and the sign bit of result is the output bit f_j. For updating block, simple multiplexing-type logic gates are enough. Table I shows the functional units used for the three major steps in different schemes. Note that the mask-and-set scheme uses the multiplexer in (9) for scoring and threshold logic gate for firing. The signals used in the generic architecture for different schemes are also listed in Table II.

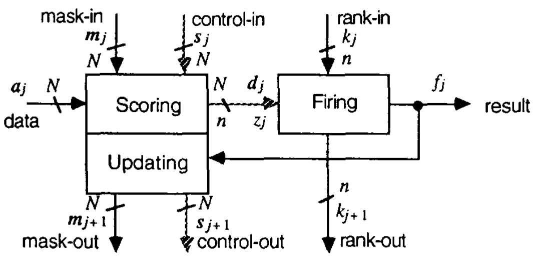

Fig. 3. A bit-sliced generic architecture for binary partition algorithms.

TABLE I
Functions Used in Different Binary Partition Schemes for the Generic Architecture with Area-Time Complexity Included

Scheme	Scoring	Firing	Updating
Counting "1"	Logic gates and counter $O(Nn)$	Subtracter $O(Nn)$	Logic gates $O(N)$
Counting "0"	Logic gates and counter $O(Nn)$	Subtracter $O(Nn)$	Logic gates $O(N)$
Counting "0" & "1"	Logic gates and counter $O(Nn)$	Sub/add $O(Nn)$	Logic gates $O(N)$
Mask+set	Logic gates $O(N)$	Threshold logic $O(N)$	Logic gates $O(N)$

TABLE II
Signals Used in the Generic Architecture for Different Binary Partition Schemes

		Mask		Control			Rank		
Scheme	Data	In	Out	In	Out	Scoring Out	In	Out	Resulting Bit
Counting "1"	a_j	m_j	m_{j+1}	—	—	z_j	k_j	k_{j+1}	f_j
Counting "0"	a_j	m_j	m_{j+1}	—	—	z_j	k_j	k_{j+1}	f_j
Counting "0" & "1"	a_j	m_j	m_{j+1}	f_{j-1}	f_j	z_j	k_j	k_{j+1}	f_j
Mask+set	a_j	m_j	m_{j+1}	s_j	s_{j+1}	d_j	—	—	f_j

B. Implementation and Tradeoffs

The implementation of a one-bit generic architecture involves a parallel counter, an adder, and some logic gates. The best area-time complexity of an n-bit adder is $AT = O(n \log n)$ [29]. The optimal area-time complexity of an N-bit parallel counter is $AT = O(N \cdot n)$ [30], where $n = \log_2 N$ and N is the window size. The hardware cost of masking and updating logic is $O(N)$ with constant time delay. Thus the total area-time complexity for realizing counting schemes would be $AT = O(N \cdot n)$.

The speed of scoring block can be improved to $O(1)$ with $O(N)$ of gates if the mask-and-set scheme is used. However, the critical problem is how to realize the threshold logic function. This function can be realized by combinational logic gates, but it costs a lot of gates when N is large. A lookup table is an alternative [25]. However, the ROM size increases exponentially with N. An interesting design of threshold gate in [26] can perform median selection in constant time with linear area complexity. Thus in the best case, the mask-and-set scheme can be realized in $AT = O(N)$. The area-time complexity is also included in Table I.

The flexibility of using counter and adder is on the changeable rank value; arbitrary rank order can be realized in the same hardware. The penalty, however, is a longer cycle time. The advantage of threshold gate design is the constant cycle time. The drawback is that the rank value is fixed at the design stage and is not modifiable during run time. When we have to determine which structure or function to take, we are trading speed of performance with flexibility during application.

C. Word-Parallel Bit-Pipeline VLSI Designs

In order to have the highest throughput rate for real-time applications, both pipeline and parallel processing techniques are needed in VLSI designs. Thus the word-parallel bit-serial generic architecture should be cascaded into pipeline stages as shown in Fig. 4. Some delay elements and delay banks are assigned at both input and output sides for skewing data bits into their correct pipeline schedules [16], [17], [21], [23]. We call this a word-parallel bit-pipeline design. It receives the whole data set in a window simultaneously and produces one result per clock cycle. The latency from dumping a window into the filter to receiving its output is L cycles.

A large number of implementations for the median filter are based on the odd–even transposition sort [16]–[18]. The basic cell is a compare-and-swap unit for a single bit. Their final sorting network architecture is also in a word-parallel bit-pipeline style. Their hardware complexity is $O(N^2L)$ and latency is $(N + L)$ pipeline cycles. The advantage of the odd–even sorting network is its good regu-

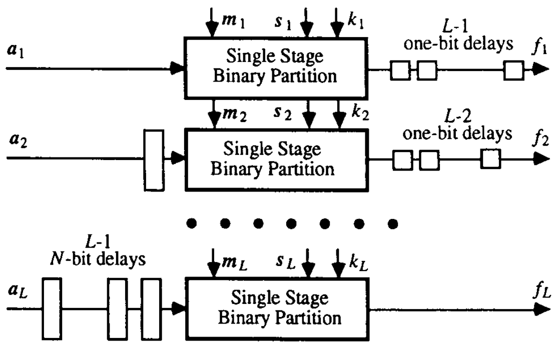

Fig. 4. The word-parallel bit-pipeline architecture for binary partition selection.

larity for VLSI implementation. However, the binary partition approach can preserve the computational efficiency while improving speed and latency. Simpler and faster implementation is possible.

V. Applications

The word-parallel bit-pipeline VLSI designs are very suitable for real-time image and signal processing applications. During the processing, all data in the working window are dumped into the filter in parallel. Though the corresponding output is generated after L cycles, the output sequence proceeds once every clock period. This is because the word-parallel bit-pipeline design can explore the maximum parallelism and meet the high throughput rate requirement of video signals.

In spite of the square window median filter, changeable window shape is also very desirable in many applications. This can be done through masking vector $\boldsymbol{m}_1 = [m_{1,1} \cdots m_{N,1}]^T$. At the initial stage, the ith element in the current window may be marked by $m_{i,1} = 1$ to indicate that it is a possible candidate. Hence we can change the length or shape of a filter window by selectively assigning "ONE" to $m_{i,1}$ in the beginning. The shape of window can be chosen as square, cross, x, or others [18], [26].

The recursive median filter is one of the major variations of standard median. It can be easily implemented by the generic architecture working in the bit-serial manner. The current output result is pulled back to the input register directly for the next selection. Neither hardware overhead nor speed degradation is observed. However, this technique is not workable for bit-pipeline designs. One possible solution is to interleave L independent image sequences into one input sequence, then the recursive operation is possible. The number of interleaved sequences depends on the number of pipeline stages or latency.

The proposed generic structure can be also applied for separable median filters or multistage median filters (median of medians). On the other hand, this generic structure can support morphological processing as well. Since the dilation and erosion of a signal or image by the structuring element is equivalent to the maximum or minimum selection in the structuring window. Using threshold logic as the firing function would be more effective because the rank value is fixed at the first or last rank. Complex morphological image processing are combinations of dilation and erosion. Thus the cascaded multistage technique is also usable.

VI. Conclusion

In this correspondence, we have unified some algorithms for ranking into a binary partition algorithmic description. This approach help us to understand the relationships between these similar algorithms and compare their advantages and drawbacks in realization. A generic architecture is proposed to implement these algorithms and consider their design issues. A word-parallel bit-pipeline design is desired for the high performance implementations. This architecture has explored the maximum parallelism of the algorithms into bit level. Binary partition algorithms have better area–time complexity than the sorting networks. The performance depends upon the realization of counter, adder, or threshold logic functions. The implementation cost may be similar, but a mask-and-set scheme may have faster speed and simpler hardware. The tradeoff is taken between the simplicity of hardware and flexibility of application. Such a binary partition approach for the implementation of rank order and median filters can be chosen to construct a customized VLSI building block in various digital image and signal applications. It can also be a component in the design library of a synthesis system for real-time video signal processing applications.

The binary partition algorithm is a very good tool in the realization of selection problems. It is believed that this approach can be applied to many other similar algorithms such as morphological filtering or nonlinear digital filters.

References

[1] G. R. Arce, N. C. Gallagher, and T. A. Nodes, "Median filters: Theory for one- and two-dimensional filters," in *Advances in Computer Vision and Image Processing*, vol. 2, *Image Enhancement and Restoration*, T. S. Huang, Ed. JAI Press, 1986, ch. 3, pp. 89-166.

[2] J. B. Bednar, "Applications of median filtering to deconvolution, pulse estimation, and statistical editing of seismic data," *Geophysics*, vol. 48, no. 12, pp. 598-1610, Dec. 1983.

[3] E. R. Ritenour, T. R. Nelson, and U. Raff, "Applications of the median filter to digital radiographic images," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1984, pp. 23.1.1-23.1.4.

[4] G. Wischermann, "Median filtering of video signals—A powerful alternative," *SMPTE J.*, pp. 541-546, July 1991.

[5] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-9, no. 4, pp. 532-550, July 1987.

[6] D. E. Knuth, *The Art of Computer Programming*, vol. 3, *Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

[7] F. Pasian, "Sorting algorithms for filters based on ordered statistics: Performance considerations," *Signal Processing*, vol. 14, pp. 287-293, 1988.

[8] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, 2nd ed. Reading, MA: Addison-Wesley, 1988.

[9] T. S. Huang, G. J. Yang, and G. Y. Tang, "A fast two-dimensional median filtering median filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, no. 1, pp. 13-18, Feb. 1979.

[10] M. O. Ahmad and D. Sundarajan, "A fast algorithm for two-dimensional median filtering," *IEEE Trans. Circuits Syst.*, vol. CAS-34, no. 11, pp. 1364-1374, Nov. 1987.

[11] J. T. Astola and T. G. Campbell, "On computation of the running median," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 572-574, Apr. 1989.

[12] M. Juhola, J. Katajainen, and T. Raita, "Comparison of algorithms for standard median filtering," *IEEE Trans. Signal Processing*, vol. 39, no. 1, pp. 204-208, Jan. 1991.

[13] S. Ranka and S. Sahui, "Efficient serial and parallel algorithms for median filtering," *IEEE Trans. Signal Processing*, vol. 39, no. 6, pp. 1462-1466, June 1991.

[14] M. O. Ahmad and D. Sundarajan, "Parallel implementation of median filtering algorithm," in *Proc. Int. Symp. Circuits Syst.*, 1988, pp. 1491-1502.

[15] A. L. Fisher, "Systolic algorithms for running order statistics in signal and image processing," *J. Digital Systems*, vol. 4, pp. 251-264, May 1984.

[16] K. Oflazer, "Design and implementation of a single chip 1-D median filter," *IEEE Trans. Acoust., Speech, Signal Processing.*, vol. ASSP-31, no. 5, pp. 1164-1168, Dec. 1983.

[17] N. Demassieux, F. Jutand, and M. Dana, "A low cost custom IC for

real-time image median in filtering," in *Proc. Custom IC Conf.*, 1986, pp. 166-169.
[18] L. A. Christopher, W. T. Mayweather, and S. S. Perlman, "A VLSI median filter for impulse noise elimination in composite or component TV signals," *IEEE Trans. Consumer Electron.*, vol. 34, no. 1, pp. 262-267, Feb. 1988.
[19] E. Ataman, V. K. Aatre, and K. M. Wong, "A fast method for real-time median filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, no. 4, pp. 415-420, Aug. 1980.
[20] P.-E. Danielsson, "Getting the median faster," *Comput. Graphics, Image Processing*, vol. 17, pp. 71-78, 1981.
[21] R. T. Hoctor and S. A. Kassam, "An algorithm and a pipeline architecture for order-statistic determination and *L*-filtering," *IEEE Trans. Circuits Syst.*, vol. 36, no. 3, pp. 344-352, Mar. 1989.
[22] M. Karaman and L. Onural, "New radix-2-based algorithm for fast median filtering," *Electron. Lett.*, vol. 25, no. 11, pp. 723-724, May 25, 1989.
[23] J. A. Roskind, "A fast sort-selection filter chip with efficiently linear hardware complexity," in *Proc. Conf. Acoust., Speech, Signal Processing*, 1985, pp. 1519-1522.
[24] B. Arambepola, "VLSI architecture for high-speed rank and median filtering," *Electron. Lett.*, vol. 24, no. 18, pp. 1179-1180, Sept. 1, 1988.
[25] K. Chen, "Bit-serial realizations of a class of nonlinear filters based on positive Boolean functions," *IEEE Trans. Circuits Syst.*, vol. 36, no. 6, pp. 785-794, June 1989.
[26] C. L. Lee and C.-W. Jen, "A bit-sliced median filter design based on majority gate," *Proc. Inst. Elec. Eng.*, pt. G, vol. 139, no. 1, pp. 63-71, Feb. 1992.
[27] P. D. Wendt, E. J. Coyle, and N. C. Gallagher, Jr., "Stack filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, no. 4, pp. 898-911, Aug. 1986.
[28] N. R. Murthy and M. N. S. Swarmy, "On the VLSI implementation of real-time order statistic filters," *IEEE Trans. Signal Processing*, vol. 40, no. 5, pp. 1241-1252, May 1992.
[29] R. B. Johnson, "The complexity of a VLSI Adder," *Inform. Process. Lett.*, vol. 11, no. 2, pp. 92-93, Oct. 20, 1980.
[30] I. Vrto, "The area-time complexity of the VLSI counter," *Inform. Process. Lett.*, vol. 25, pp. 397-400, July 26, 1987.
[31] S. Muroga, *Threshold Logic and Its Applications*. New York: Wiley, 1971.

Uniqueness of a Two-Step Predictor Based Spectral Estimator that Generalizes the Maximum Entropy Concept

Theodore I. Shim, S. Unnikrishna Pillai, and Won Cheol Lee

***Abstract*—Given a finite set of autocorrelations, it is well known that maximization of the entropy functional subject to this data leads to a stable autoregressive (AR) model. Since maximization of the entropy functional is equivalent to maximization of the minimum mean square error associated with one-step predictors, the problem of obtaining admissible extensions that maximize the *k*-step minimum mean square prediction error subject to the given autocorrelations is meaningful, and it has been shown to result in stable ARMA extensions (see the work by Pillai *et al.*). The uniqueness of this true generalization of the maximum entropy extension is proved here through a constructive procedure in the case of two-step predictors.**

Manuscript received February 19, 1992; revised October 2, 1992. The associate editor coordinating the review of this correspondence and approving it for publication was Prof. Georgios B. Giannakis. This work was supported by the National Science Foundation under Contract MIP-9020501 and the Office of Naval Research under Contract N-00014-89-J-1512.

The authors are with the Department of Electrical Engineering, Polytechnic University, Brooklyn, NY 11201.

IEEE Log Number 9210111.

I. Introduction

Given a finite set of autocorrelations $r_0, r_1, \cdots, r_n$, from a zero-mean wide-sense stationary discrete-time stochastic process, the solution to the problem of obtaining the entire class of spectral extensions that match the given autocorrelations subject to certain mild restrictions is well known [2]-[4]. Among the infinite possibilities that include both rational as well as nonrational extensions, the particular one that maximizes the entropy functional results in a stable AR(n) model that is uniquely specified by the Levinson polynomial associated with the given autocorrelations [5]. Since maximization of the entropy functional is equivalent to maximization of the minimum mean square prediction error associated with one-step predictors that make use of the entire past data samples [6], the general problem of obtaining spectral extensions that maximize the multistep minimum mean square prediction error is meaningful. In this context, it has been shown that [1], given the autocorrelations $r_0, r_1, \cdots, r_n$, maximization of the k-step minimum mean square prediction error results in stable ARMA$(n, k - 1)$ extensions, and in this sense, this is a direct generalization of the maximum entropy solution. Further, details of this particular extension have been worked out in the case of two-step predictors ($k = 2$) by making use of the general formulation regarding the class of all extensions, and it was shown that at most two distinct minimum phase extensions exist that could maximize the two-step minimum mean square prediction error [1].

In this correspondence, we proceed to show that, in fact, there is always one and only one extension that maximizes the two-step minimum mean square prediction error and the details of this unique minimum phase ARMA$(n,1)$ extension are worked out here.

II. Uniqueness of an ARMA$(n,1)$ Extension

Consider a discrete-time zero-mean wide-sense stationary stochastic process $x(nT)$ with autocorrelation coefficients $r_k = E[x(nT)x^*((n + k)T)] = r^*_{-k}$, $k = 0 \rightarrow \infty$ and power spectral density

$$S(\theta) = \sum_{k=-\infty}^{+\infty} r_k e^{jk\theta} \geq 0$$

that satisfies the integrability condition

$$r_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} S(\theta)\, d\theta < \infty \tag{1}$$

and the causality criterion [3], [4], [6]

$$H = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln S(\theta)\, d\theta > -\infty. \tag{2}$$

It is well known that [6] such a power spectral density function can be factorized in terms of its unique Wiener factor $B(z)$ such that $S(\theta) = |B(e^{j\theta})|^2$ *a.e*, where

$$B(z) = \sum_{k=0}^{\infty} b_k z^k, \qquad b_0 > 0 \tag{3}$$

is analytic together with its inverse in $|z| < 1$ (minimum phase factor that is free of zeros in $|z| < 1$) and it satisfies the square summability condition $\Sigma_{k=0}^{\infty} |b_k|^2 < \infty$. Moreover, the constant term in the Wiener factor is related to the entropy of the process through the relation [3], [6]

$$|b_0|^2 = \exp\left(\frac{1}{2\pi} \int_{-\pi}^{\pi} \ln S(\theta)\, d\theta\right) = \exp(H). \tag{4}$$