High Sample Rate Array Architectures for Median Filters

Chaitali Chakrabarti

Department of Electrical Engineering
Center for Solid State Electronics Research
Arizona State University
Tempe, AZ 85287-5706
(602) 965-9516
email: chaitali@enuxha.eas.asu.edu

Abstract

This paper presents high sample rate semi-systolic array architectures for computing 1-D and 2-D non-recursive and recursive median filters. A high sample rate is obtained by pipelining the computations in each processor. While the non-recursive filters are pipelined by placing latches in the feed-forward paths, the recursive filters are restructured to create additional delays in the feedback paths, and then pipelined using the delays as latches.

EDICS 7.1.1, 7.1.4

1 Introduction

The median filter is a non-linear digital filter with an interesting combination of properties: it can reduce high frequency and impulsive noises without destroying edge information. These properties make it a very popular filter in speech processing and image processing schemes.

There are two types of median filters: non-recursive and recursive. In 1-D non-recursive median filtering, a window of size $(2N+1)$ is moved across the entire signal, and the center sample of each window is replaced by the median of the samples in the window. The idea is that the median of the samples in the window is the best guess for the center sample. Let W_i be a window centered on the ith sample, that is, $W_i = \{x_{i-N}, \ldots, x_i, \ldots, x_{i+N}\}$. Then y_i is the median of the samples in W_i, $y_i = median\{W_i\}$. In 2-D median filtering, a window of size $(K \times K)$ is moved across the image, $K = 2N+1$. Let $W_{i,j}$ be the window centered on (i,j), then $y_{i,j} = median\{W_{i,j}\}$. In recursive median filtering, the window consists of the recent median samples as well as the input samples. For instance, the ith window for 1-D recursive filtering $\tilde{W}_i = \{y_{i-N}, \ldots, y_{i-1}, x_i, \ldots, x_{i+N}\}$, and $y_i = median\{\tilde{W}_i\}$. By including the recent median samples, the features from the input signal can be extracted faster than by non-recursive median filters [8]. Rank order filters are generalized form of median filters where the output is the element with rank R, that is, the output is the Rth smallest element. Thus a median filter with window size $2N+1$ is a rank order filter with $R = N+1$.

The existing architectures for median filters can be broadly classified into two classes [1]: ones in which the samples are stored in order of arrival, and others in which samples are stored in sorted order. There are other stack filter based architectures where the median is obtained by examining the bits of the samples [4]. [11] gives an excellent survey of the existing architectures.

The architectures belonging to the first class (samples stored in order of arrival) are either sorting network based architectures [9, 2] or are linear systolic array architectures [7, 6, 3, 10]. While the sorting networks can be pipelined to any level, they consist of at least $O(K \log^2 K)$ compare-swap units for window size K, $K = 2N+1$. The linear systolic array architectures, on the other hand, consist of only $2N+1$ processors, but have a larger sample period. In the linear array architectures, the sample period is a function of the time taken to update the ranking of the old window based on comparison results with the new and oldest samples. The architectures for the case when samples are stored in sorted order [1, 5] also consist of a linear array of $2N+1$ processors. Here the sample period is a function of the time taken to update the sorted list by discarding the oldest element and inserting the new element. In [3] we developed array architectures with lower sample periods than existing array implementations for the case when samples are stored in order of arrival as well as when samples are stored in sorted order.

In this paper we extend our previous results [2, 3] and develop high sample rate architectures for 1-D and 2-D non-recursive and recursive median filters. The architectures consist of a semi-systolic array of processors which store the samples in order of arrival. We first describe non-pipelined designs for 1-D and 2-D non-recursive median filters with lower sample periods than existing architectures [6, 7, 10], and then show how to pipeline these designs to any level by placing latches in the feed-forward paths. We next describe a design with low sample period for 1-D recursive median filters. We show how to pipeline the design by restructuring the algorithm to create additional delays in the feedback paths to be used as pipeline latches. In Sections 2, 3, and 4, we describe the architectures and algorithms for 1-D and 2-D non-recursive median filters, and 1-D recursive median filters.

2 1-D non-recursive median filters

In this section we describe a highly pipelined semi-systolic linear array architecture for 1-D non-recursive median filtering. The algorithm is based on the fact that ranking of the present window can be obtained by updating the ranking of the old window (rather than re-evaluation from scratch) [7]. The systolic linear array architectures proposed by Kung [7] and Hwang [6] obtain the ranking of the present window by updating the ranking of the old window twice; once by comparing with the sample being discarded and once by comparing with the new sample. The sequential update of the ranks, and the purely systolic nature of the design, makes the sample period of the designs in [6, 7] quite large (see Table I).

We propose an architecture consisting of a semi-systolic linear array of $(2N+1)$ processors (see Figure 1) which have a lower sample period than the existing architectures [6, 7, 10]. The processors store the samples in the order of arrival; P_0 stores the new sample and P_{2N} the oldest sample. Each processor updates its rank based on the comparison results with the new and the discarded samples. Processor P_0 computes the rank of the new sample by counting the number of samples which are smaller than its own. All processors (except P_0) consist of (i) data register a to store the sample value, (ii) rank register r to store the rank of that sample, (iii) a comparator to compare with the new sample, (iv) a shift register s to store the comparison results with all samples that arrived before it, (v) rank update unit to update the old rank, and (vi) an output unit to output the median value. By storing the comparison results with older samples in register s, each processor no longer needs to compare with the discarded sample. The comparison result with the discarded sample is in the rightmost cell of register s. Figure 2 describes the design of such a processor. Processor P_0 consists of registers a, r, s, a carry look-ahead adder tree and an output unit.

We next describe the operations that take place during the computation of the ith output. For the ith window, the new sample $x_{new} = x_{i+N}$, and the discarded sample $x_{old} = x_{i-N-1}$.

Step 1. Compare with new sample: Each processor (except P_0) compares the contents of its register a with x_{new}, and stores the comparison result in 1-bit register u; $u = 1$ if $a \geq x_{new}$[1]. The rightmost cell of register s contains the comparison result with x_{old}. This is loaded into 1-bit register v.

Step 2.1. Update rank: Each processor, except P_0, updates its rank register r, $r = r + u - v$.

Step 2.2. Compute rank of new sample: Processor P_0 computes the rank of x_{new} by counting the number of samples that are less than x_{new}. This is achieved by computing $r = 1 + \sum_{j=1}^{2N} \bar{u}(P_j)$ by a carry look-ahead adder tree.

Step 2.3. Update comparisons with other samples: All processors, except P_0, shift their register s right. Processor P_0 loads the contents of $\bar{u}(P_i)$ into $s[i]$, $1 \leq i \leq 2N$.

Step 3. Match rank: Any processor whose rank matches $R = N + 1$, outputs the contents of data register a. For rank order filters, R is set to the rank of the filter. Registers a, r, and s of processor P_j are shifted to processor P_{j+1}, $0 \leq j < 2N$.

The sample period of our design is given by $T_{sample} = T_{comp}(n) + max\{T_{add}(b), T_{cla}(2N+1)\} + T_{comp}(b)$, where n is the number of bits to represent a sample value, b is the number of bits to represent rank of the sample ($b = \lceil \log(2N+1) \rceil$), $T_{comp}(s)$ is the time required to compare two s bit numbers, $T_{add}(s)$ is the time required to add two s bit numbers, and $T_{cla}(s)$ is the time required to add s 1-bit numbers using a tree of carry look-ahead adders. Figure 4 describes the above sequence of operations for median computation in an example with window size 5.

Recently Lucke and Parhi [10] independently proposed an architecture for 1-D median filtering. In their architecture [10], the rank of the new element is computed first, and the ranks of the remaining elements are updated by comparing with the ranks of the new and the oldest elements. In comparison, in our architecture [3], the rank of the new element is computed in parallel with the rank updates of the remaining elements, resulting in a lower sample period. Table I compares the sample period and the hardware requirements of our architecture with those in [10] and [6].

Pipelining: The sample period of our design can be reduced by pipelining the computations in each processor. For example, the sample period can be reduced by a factor of 3 by adding latches in the feed-forward paths as shown in Figure 2. The trade-off is an increase in area (due to the latches) and an increase in latency. Figure 3 describes an alternate implementation which results in fewer pipeline registers. Here the samples of the window are stored in a centralized register bank of size $2N + \alpha$, for α levels of pipelining. Now if in Step 3 of the procedure, processor P_i has rank $N + 1$,

[1] Since $u = 1$ if $a \geq x_{new}$, the rank of x_{new} is guaranteed to be less than that of any other window element with the same value. This also implies that the ranks of all elements in the window are distinct.

then the contents of the $(i+\alpha-1)$th register is sent to the output bus.

3 2-D non-recursive median filters

The linear array architecture proposed for 1-D median filtering can be easily extended to compute 2-D median filtering as in [6, 10]. The array consists of K^2 processors, and a new output is determined *every* K cycles. In this section we first propose a high-throughput architecture for 2-D median filtering where an output is determined *every* cycle, and then show how this architecture can be modified for the case when an output is determined *every* T cycles, $1 \leq T \leq K$. The area complexity of the high throughput architecture is $O(K^3 n)$ compared to $O(K^2 \log^2 Kn)$ for odd-even merge sorting network based architectures.

The proposed architecture consists of a $(K \times K)$ array of processors, where processors P_0 through P_{K-1} (in the leftmost column) store K samples of the new column (see Figure 5). Processors P_K through P_{K^2-1} update their ranks based on the comparison results with the K new samples and the K discarded samples, while processors P_0 through P_{K-1} compute their ranks by counting the number of samples that are smaller than their sample values. Processors P_K through P_{K^2-1} contain K comparators to compare with the K new samples, data register a, rank register r, register array s [2] to store the comparison results with all samples that arrived before it, rank update unit and an output unit. Processors P_0 through P_{K-1} contain $(K-1)$ comparators, registers a and r, register array s of size $(K \times K-1)$, a carry look-ahead adder tree, and an output unit.

The operations for the computation of the median of a 2-D window are summarized as follows.

Step 1. Compare with K new samples: Each processor, except P_0 through P_{K-1}, compares the K new samples with the contents of its register a, and stores the comparison results in 1-bit registers $\{u_m\}$, $0 \leq m \leq K-1$. The cells in the rightmost column of the register array s contain the comparison results with the discarded samples. These are loaded into 1-bit registers $\{v_m\}$.

Step 2.1. Update rank: Each processor, except P_0 through P_{K-1}, updates its rank register by $r = r + \sum_{m=0}^{K-1} u_m - \sum_{m=0}^{K-1} v_m$.

Step 2.2. Compute rank of K new samples: Each processor P_m, $0 \leq m < K$, computes the rank of its sample by a tree of carry look-ahead adders, $r_m = 1 + \sum_{\substack{i=0 \\ i \neq m}}^{K^2-1} \bar{u}_m(P_i)$.

Step 2.3. Update comparisons with other samples: Each processor P_K through P_{K^2-1} shifts its register array s right. Processor P_m, $0 \leq m < K$, loads the contents of $\bar{u}_m(P_i)$ into s, $K \leq i \leq K^2-1$.

Step 3. Match rank: Any processor whose rank matches $R = \lceil K^2/2 \rceil$ outputs the contents of

[2]The size of the register array decreases from $(K \times K-1)$ in the leftmost column of processors to $(K \times 1)$ in the rightmost column of processors.

register a. Registers a, r, and s of processor P_j are shifted to processor P_{j+K}, $0 \leq j < K^2 - K$.

The sample period of this design is $T_{sample} = T_{comp}(n) + max\{T_{cla}(2K) + T_{add}(b), T_{cla}(K^2)\} + T_{comp}(b)$. The sample period can be reduced by pipelining the computations in each processor. As in the case of 1-D filtering, pipelining is achieved by placing latches in the feed-forward paths. Figure 6 traces the above sequence of operations for 2-D median computation in an example with a (3×3) window. The new column of samples is $(3, 2, 0)^T$, and the discarded column of samples is $(10, 3, 1)^T$.

Area-time trade-off

The above architecture can be generalized for the case when an output is determined every T sample periods, $1 \leq T \leq K$. The number of processors is still K^2, however, now in 1 sample period, the ranks of K/T new elements are determined, and the ranks of $K^2 - K/T$ samples are updated. Each processor now consists of only K/T comparators to compare with the K/T new elements. The number of carry look-ahead adders is also reduced to K/T. Thus when the sample period is increased by a factor of T, the area of the resulting processor array is reduced to $O(K^3 n/T)$.

4 1-D recursive median filters

The window $\tilde{W}_i$ for 1-D recursive median filtering consists of the last N output samples, y_{i-1} through y_{i-N}, and $(N+1)$ input samples, x_i through x_{i+N}; $\tilde{W}_i = \{y_{i-N}, \ldots, y_{i-1}, x_i, \ldots, x_{i+N}\}$ and $y_i = median\{\tilde{W}_i\}$. In this section we first describe a semi-systolic linear array architecture for 1-D recursive median filtering without pipelining, and then show how this architecture can be pipelined to achieve high sample rates.

4.1 Architecture without pipelining

The architecture that we propose for 1-D recursive median filtering consists of $(2N+1)$ processors arranged in 2 rows as shown in Figure 7. The first row stores the N y-samples in processors Q_0 through Q_{N-1}, and the second row stores the $(N+1)$ x-samples in processors P_0 through P_N. The output and the input samples are stored in the order of arrival. In every sample period, 2 new samples are written into the leftmost processors of the array, and 2 samples are discarded from the rightmost processors of the array (see Figure 7). All processors (except P_0) update their ranks based on the comparison results with the 2 new and the 2 discarded samples. Processor P_0 computes its rank by counting the number of sample values that are smaller than its own. Figure 8 illustrates the design of a single processor. All processors except P_0 [3] consist of a comparator, data register a, rank register r, register s_y to store comparisons with all the y-samples, register s_x to store comparisons with all the x-samples, rank update unit and an output unit.

[3] Processor P_0 contains registers a, r, s_x, s_y, a carry look-ahead adder tree and an output unit.

The operations for 1-D recursive median computation are similar to those for 2-D non-recursive median computation for the case where 2 new elements are input every sample period. The operations for the computation of an output are summarized as follows. Let x_{new} and y_{new} be the two new samples and let x_{old} and y_{old} be the two discarded samples. For the ith window, $x_{new} = x_{i+N}$, $y_{new} = y_{i-1}$, $x_{old} = x_{i-1}$ and $y_{old} = y_{i-N-1}$.

Step 1. Compare with new x-sample: Each processor, except $P(0)$, compares the contents of register a with x_{new}, and stores the comparison result in 1-bit register u_1; $u_1 = 1$ is $a \geq x_{new}$. The comparison results with y_{new} is stored in register u_0; $u_0 = 1$ if $a > y_{new}$. u_0 is set in Step 3 of this procedure. The comparison results with y_{old} and x_{old} are stored in the rightmost cells of registers s_y and s_x, and are loaded into 1-bit registers v_0 and v_1; $v_0 = s_y[N-1]$, and $v_1 = s_x[N]$.

Step 2.1. Update rank: Each processor, except P_0, updates its rank register r by $r = r + u_0 + u_1 - v_0 - v_1$.

Step 2.2. Compute rank of new sample: Processor P_0 computes its rank by a tree of carry look-ahead adders, $r = 1 + \sum_{i=0}^{N-1} \bar{u}_0(Q_i) + \sum_{j=1}^{N} \bar{u}_1(P_j)$.

Step 2.3. Update comparisons with other samples: All processors, except P_0, shift their registers s_x and s_y right, and load u_1 into the leftmost cell of s_x and u_0 into the leftmost cell of s_y. P_0 loads the contents of $\bar{u}_0(Q_i)$ into $s_y[i]$, $0 \leq i \leq N-1$, and the contents of $\bar{u}_1(P_j)$ into $s_x[j]$, $0 \leq j \leq N$.

Step 3. Match rank: Each processor compares its rank r with $R = N + 1$. If $r > R$, then it sets $u_0 = 1$, otherwise it sets $u_0 = 0$. If $r = R$, then it gates the contents of its register a to the output bus, and the contents of registers a, r, s_x and s_y to processor Q_0. All processors transfer the contents of register a, r, s_x, s_y, u_0 to their right adjacent processors.

The sample period of our architecture is given by $T_{sample} = T_{comp}(n) + max\{T_{cla}(4) + T_{add}(b), T_{cla}(2N+1)\} + T_{comp}(b)$.

The recursive median filter architecture recently proposed by Lucke and Parhi [10] has a larger sample period compared to our architecture. Their algorithm consists of (i) computing rank of x_{new}, (ii) using rank of x_{new} to compute rank of y_{new}, (iii) updating ranks of the remaining $2N-1$ elements by +2,+1,0,-1,-2 depending on ranks of y_{old}, x_{old}, y_{new}, x_{new} and the current rank, and (iv) outputting median. The complexity of the rank update logic (in Step iii) and the sequential computation of Steps (i), (ii) and (iii) makes their design more complex compared to ours. Table II compares the sample period and hardware requirements of both the architectures.

4.2 Architecture with pipelining

Recursive median filters cannot be pipelined by only adding pipeline latches in the feedforward paths (as in non-recursive median filters). For instance, if a set of latches is added to the feed-forward path of the processor in Section 4.1, the processor array would implement the function $y_i = median\{y_{i-N-1}, \ldots, y_{i-2}, x_i, \ldots, x_{i+N}\}$ instead of $y_i = median\{y_{i-N}, \ldots, y_{i-1}, x_i, \ldots, x_{i+N}\}$. In this section we describe modifications in the recursive median filtering algorithm in order to incorporate pipelining. We first describe an architecture with 2 levels of pipelining, and then show how this idea can be extended for higher levels of pipelining.

4.2.1 Pipelining to 2 levels

In order to pipeline the recursive median filter design to 2 levels, the algorithm for the computation of y_i has to be modified in such a way that y_i is a function of the output samples y_{i-2} through y_{i-N-1}, instead of y_{i-1} through y_{i-N}. Let $\hat{W}_i$ be a window of size $(2N+2)$ consisting of N output samples, $y_{i-N-1}, \ldots, y_{i-2}$, and $(N+2)$ input samples, $x_{i-1}, \ldots, x_{i+N}$, $\hat{W}_i = \{y_{i-N-1}, \ldots, y_{i-2}, x_{i-1}, \ldots, x_{i+N}\}$. Recall that $W_i = \{y_{i-N}, \ldots, y_{i-1}, x_i, \ldots, x_{i+N}\}$. We propose an algorithm to compute $y_i = median\{W_i\}$ by operating on window $\hat{W}_i$. It can be shown that y_i is the element with rank $(N+1)$ or rank $(N+2)$ in $\hat{W}_i$, depending on the ranks of x_{i-1}, y_{i-N-1} and x_{i+N} in $\hat{W}_i$. A brief sketch of the correctness of the algorithm is included in the Appendix. Let e_{N+1} be the element with rank $N+1$ in $\hat{W}_i$, and let e_{N+2} be the element with rank $N+2$ in $\hat{W}_i$.

Algorithm:

If $(r(x_{i-1}) \leq N+1)$

then {

if $((r(x_{i+N}) > N+1)$ *and* $(r(y_{i-N-1}) > N+1))$

then $y_i = e_{N+1}$

else $y_i = e_{N+2}$.

}

else {

if $((r(x_{i+N}) < N+2)$ *and* $(r(y_{i-N-1}) < N+2))$

then $y_i = e_{N+2}$.

else $y_i = e_{N+1}$.

}

We propose an architecture consisting of $(2N+2)$ processors for recursive median filtering with 2 levels of pipelining. The processors are arranged in 2 rows, such that the first row stores the N y-samples, and the second row stores the $(N+2)$ x-samples. The output and the input samples are stored in the order of arrival. Thus processors Q_0 and P_0 store the two newest samples and processors Q_{N-1} and P_{N+1} store the two oldest samples of the window.

The operations for the pipelined recursive median computation are very similar to the non-pipelined case. The following operations are performed during the computation of y_i:
In Step 1 (compare with new sample), each processor, except P_0, compares the contents of its data register a with x_{new} and stores the comparison results in u_1. The comparison results with y_{old} and x_{old} are loaded from the rightmost cells of s_y and s_x, and are stored in v_0 and v_1 respectively. The comparison result with y_{new} is set in Step 3, and is stored in u_0.
In Steps 2.1 and 2.2 (rank computation), processor P_0 computes its rank by $r_0 = 1 + \sum_{i=0}^{N-1} \bar{u}_0(Q_i) + \sum_{j=1}^{N+1} \bar{u}_1(P_j)$, and the remaining processors update their ranks depending on the values of u_0, u_1, v_0 and v_1 registers. In Step 2.3 all the processors update their s_x and s_y registers.
In Step 3 (match rank), all processors compare their ranks with ranks $(N+1)$ and $(N+2)$. The ranks of x_{i-1} in P_{N+1}, x_{i+N} in P_0, and y_{i-N-1} in Q_{N-1}, are used to decide whether y_i is the element with rank $(N+1)$ or rank $(N+2)$. Let $e=0$ if $r(x_{i-1}) \leq N+1$, $f=0$ if $r(x_{i+N}) \leq N+1$, and $g=0$ if $r(y_{i-N-1}) \leq N+1$. $y_i = e_{N+1}$ if $fg + e(f+g) = 1$, *else* $y_i = e_{N+2}$. u_0 is set to 1 if *either* $r > N+2$ *or* if $(r = N+2)$ *and* $(fg + e(f+g) = 1)$.

y_i is fed back to the processor array for the computation of y_{i+2}. Figure 9 traces the computations for an example with window size 5. $\hat{W}_i = \{9, 9, 12, 2, 5, 6\}$, $x_{old} = 10$ and $y_{old} = 9$. In order to reduce the sample period of the proposed architecture by a factor of 2, latches could be added in the rank update unit in all processors (except P_0), and in the carry look-ahead adder tree in processor P_0.

4.2.2 Pipelining to 3 levels

In order to pipeline the recursive filter to 3 levels, y_i should be a function of N outputs y_{i-3} through y_{i-N-2}. Let $\tilde{W}_i$ be a window of size $(2N+3)$ such that $\tilde{W}_i = \{y_{i-N-2}, \ldots, y_{i-3}, x_{i-2}, \ldots, x_{i+N}\}$. It can be shown that $y_i = median\{W_i\}$ is rank $(N+1)$ or rank $(N+2)$ or rank $(N+3)$ of $\tilde{W}_i$ depending on the ranks of y_{i-N-2}, y_{i-N-1}, x_{i-2}, x_{i-1}, x_{i+N-1} and x_{i+N}. An implementation of this algorithm results in a complex output unit, making reduction of the sample period by a factor of 3 impossible.

In general, in order to pipeline the recursive filter to M levels, y_i should be a function of N outputs, y_{i-M} through $y_{i-N-M+1}$, and $(N+M)$ inputs, x_{i-M+1} through x_{i+N}. For $M > 2$, the *match rank* step of the algorithm becomes increasingly complex, and the delay in the output unit dominates the delay of the design.

5 Conclusion

In this paper we have described high sample rate array architectures for computing non-recursive and recursive median filters. We have developed non-pipelined architectures with lower sample periods than existing array architectures, and then reduced the sample periods further by pipelining the processors. The non-recursive filters have been pipelined by placing latches in the feed forward paths. The trade-off was in the increase in area (due to pipeline latches) and the increase in latency. The recursive filters could not be pipelined as easily. The recursive algorithm was restructured to create additional delays in the feedback paths, which could be used as pipeline latches. The complexity of the algorithm increased considerably with the increase in the number of levels of pipelining.

Acknowledgements: The author would like to thank Prof. Lori Lucke for suggesting the implementation described in Figure 3, and the reviewers for their insightful comments.

References

[1] G. R. Arce, P. J. Warter, "A median filter architecture suitable for VLSI implementation," Proc. of 23rd Annual Allerton Conf. on Comm., Control, and Computing, 1984, pp. 172-181.

[2] C. Chakrabarti, "High Sample Rate Architectures for Median Filters," VLSI Signal Processing V, IEEE Press, pp. 490-499.

[3] C. Chakrabarti, "High Sample Rate Systolic Architectures for Median Filters," Proc. IEEE Int. Symp. on Circuits and Systems, May 1992, pp. 1073-1076.

[4] J. P. Fitch, E. J. Coyle, and N. C. Gallagher, "Median Filtering by Threshold Decomposition," IEEE Trans on Acoust., Speech, and Signal Process., vol. 32, 1984, pp. 1183-1188.

[5] A. L. Fisher, "Systolic algorithms for running order statistics in signal and image processing," J. Digital Syst., vol. 4, 1982, pp. 251-264.

[6] J. N. Hwang, J. M. Hong, "Systolic architectures for 2-D rank order filtering," Proc. Int. Conf. Application Specific Array Processors (Princeton, NJ), 1990, pp. 90-99.

[7] S. Y. Kung, "VLSI Array Processors," Prentice Hall, 1989.

[8] T. A. Nodes, N. C. Gallager, "Median filters: Some modifications and their properties," IEEE Trans on Acoust., Speech, and Signal Processing, vol. 30, 1982, pp. 739-746.

[9] K. Oflazer, "Design and implementation of a single chip 1-D median filter," IEEE Trans on Acoust., Speech, and Signal Processing, vol. 31, 1983, pp.1164-1168.

[10] L. E. Lucke, K. K. Parhi, "A New VLSI Architecture for Rank Order and Stack Filters," Proc. IEEE Int. Symp. on Circuits and Systems, May 1992, pp. 101-104.

[11] D. Richards, "VLSI median filters," IEEE Trans on Acoust., Speech, and Signal Processing, vol. 38, 1990, pp. 145-153.

Appendix

The algorithm in Section 4.2.1, shows that y_i = rank $(N+1)$ of W_i is the element with rank $(N+1)$ or $(N+2)$ of $\hat{W}_i$, depending on the ranks of x_{i-1}, x_{i+N}, and y_{i-N-1} in $\hat{W}_i$. The correctness of this algorithm can be proved by analyzing all possible cases. Let the elements with ranks $N+1$ and $N+2$ in $\hat{W}_i$ be denoted by e_{N+1} and e_{N+2} respectively.

Case 1: If $r(x_{i-1}) \leq N+1$ *and*

i) $r(y_{i-N-1}) < r(x_{i-1}) < r(x_{i+N})$, *then* $y_i = e_{N+2}$

ii) $r(x_{i+N}), r(y_{i-N-1}) < r(x_{i-1})$, *then* $y_i = e_{N+2}$

iii) $r(x_{i+N}) < r(x_{i-1}) < r(y_{i-N-1})$, *then* $y_i = e_{N+2}$

iv) $r(x_{i-1}) < r(x_{i+N}), r(y_{i-N-1})$, *and*

if $(r(x_{i+N}) > N+1$ *and* $r(y_{i-N-1}) > N+1)$ *then* $y_i = e_{N+1}$; *else* $y_i = e_{N+2}$.

Case 2: If $r(x_{i-1}) \geq N+2$ *and*

i) $r(y_{i-N-1}) < r(x_{i-1}) < r(x_{i+N})$, *then* $y_i = e_{N+1}$

ii) $r(x_{i+N}), r(y_{i-N-1}) < r(x_{i-1})$, *and*

if $(r(x_{i+N}) < N+2$ *and* $r(y_{i-N-1}) < N+2)$ *then* $y_i = e_{N+2}$; *else* $y_i = e_{N+1}$.

iii) $r(x_{i+N}) < r(x_{i-1}) < r(y_{i-N-1})$, *then* $y_i = e_{N+1}$

iv) $r(x_{i-1}) < r(x_{i+N}), r(y_{i-N-1})$, *then* $y_i = e_{N+1}$.

Let the set of elements in $\hat{W}_i$ that have ranks smaller than x_{i-1} be denoted by $\{S_1\}$, and the set of elements that have ranks greater than x_{i-1} be denoted by $\{S_2\}$. Let l be the rank of x_{i-1}. Then $\{S_1\}$ contains $(l-1)$ elements and $\{S_2\}$ contains $(2N-l+2)$ elements. The proofs of all these cases consist of first determining the rank of y_{i-1} in $\{S_1\}$ or $\{S_2\}$, and then using that information to determine the rank of y_i in $\{S_1\}$ or $\{S_2\}$. To illustrate the procedure we prove case 1i.

Proof of Case 1i :

Given that $r(x_{i-1})$ is $l \leq N+1$, and that $r(y_{i-N-1}) < r(x_{i-1}) < r(x_{i+N})$, to prove that y_i is the element with rank $N+2$ in $\hat{W}_i$.

Since $W_{i-1} = \hat{W}_i - x_{i+N}$, and $l \leq N+1$, $y_{i-1} = median\{W_{i-1}\}$ is the $(N+1-l)$th smallest element of $\{S_2 - x_{i+N} + x_{i-1}\}$. Thus y_{i-1} is the $(N+1-l)$th smallest element of $\{S_2 + x_{i-1}\}$ if $x_{i+N} > y_{i-1}$, and is the $(N+2-l)$th smallest element of $\{S_2 + x_{i-1}\}$ if $x_{i+N} < y_{i-1}$.

W_i consists of elements in the set $\{S_1 - y_{i-N-1} + S_2 + y_{i-1}\}$. Since the elements in the set $\{S_1 - y_{i-N-1}\}$ are less than the elements in the set $\{S_2 + y_{i-1}\}$, and the number of elements in the set $\{S_1 - y_{i-N-1}\}$ is $l-2$, y_i lies in the set $\{S_2 + y_{i-1}\}$. In fact, y_i is the $(N+1-(l-2)) = (N+3-l)$th smallest element of $\{S_2 + y_{i-1}\}$ which is the $(N+2-l)$th smallest element of S_2. Thus y_i is the $((N+2-l)+l) = (N+2)$th smallest element of $\hat{W}_i$.

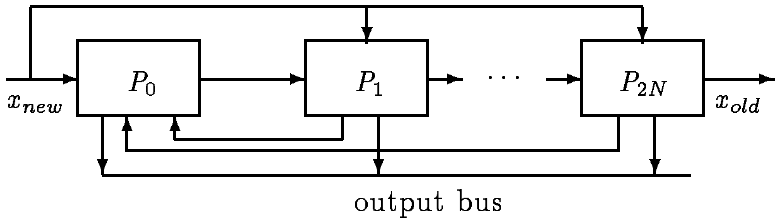


Figure 1: Linear array architecture for 1-D non-recursive median filtering

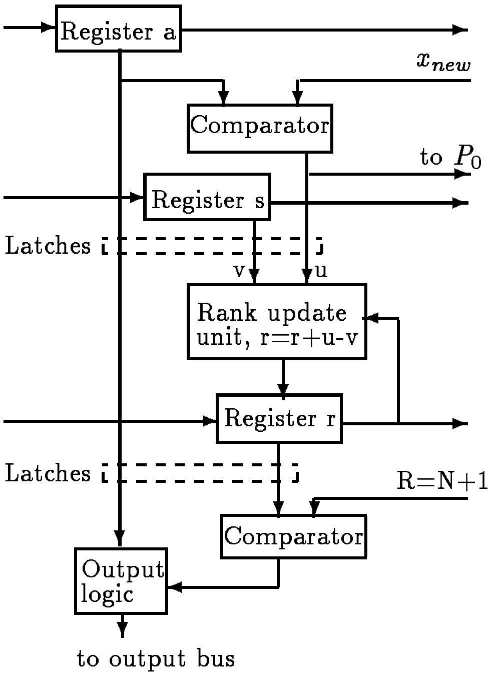


Figure 2: Block diagram of a processor for 1-D non-recursive median filtering pipelined to 3 levels

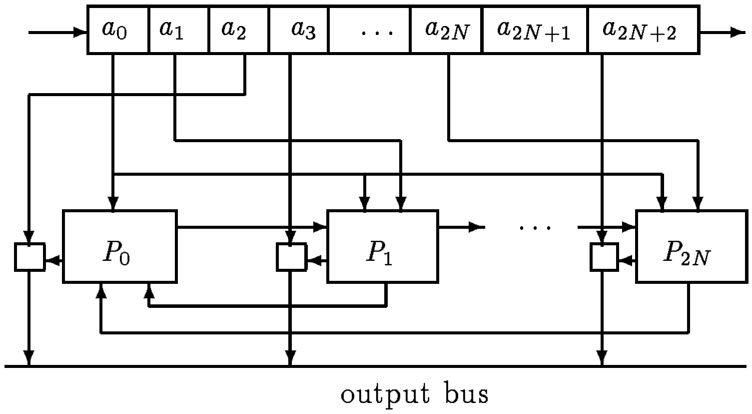


Figure 3: Alternate linear array architecture for 1-D non-recursive median filtering pipelined to 3 levels

W_{i-1}={6,8,5,2,3}

W_i={2,6,8,5,2} x_{new} of W_i is 2, x_{old} of W_i is 3

	P_0	P_1	P_2	P_3	P_4
Step 1:					
Register a	2	6	8	5	2
Register s	-	0111	111	11	0
Comp. with x_{old}, v	-	1	1	1	0
Comp. with x_{new}, u	-	1	1	1	1
Step 2:					
Register r	-	5	4	3	1
Update r	1	5	4	3	2
Update s	0000	011	11	1	-
Step 3:					
Comp. with R=3				Match	
Output				5	

Figure 4: Sequence of operations for 1-D non-recursive median filtering

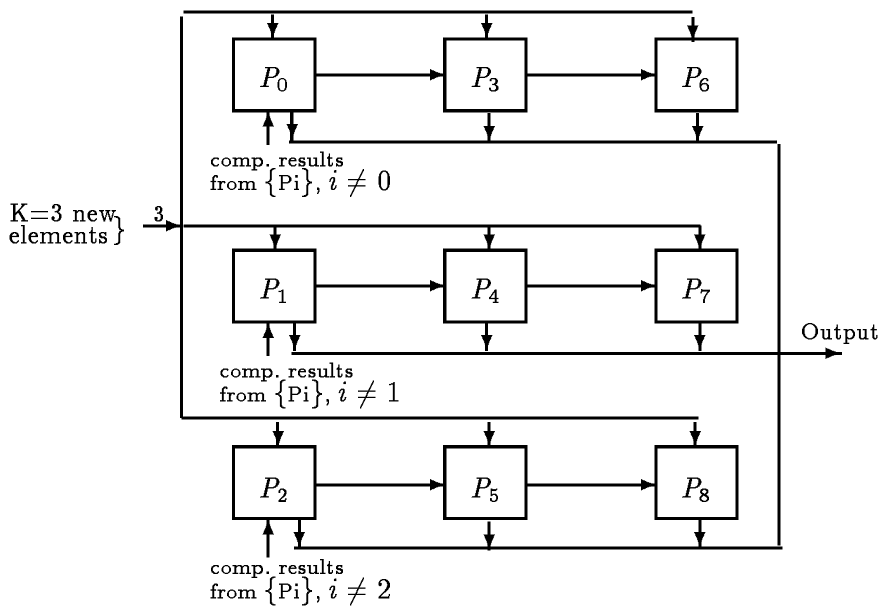
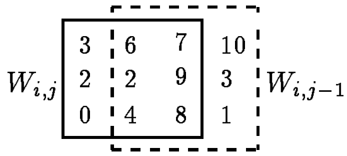


Figure 5: Array architecture for 2-D non-recursive median filtering when the window size is (3×3)



	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Step 1:									
Register a	3	2	0	6	2	4	7	9	8
Register s	-	-	-	00	00	00	0	0	0
	-	-	-	01	00	01	1	1	1
	-	-	-	01	01	01	1	1	1
Comp. with u_1	-	0	0	1	0	1	1	1	1
new elements u_2	1	-	0	1	1	1	1	1	1
3,2,0 u_3	1	1	-	1	1	1	1	1	1
Comp. with v_1	-	-	-	0	0	0	0	0	0
old elements v_2	-	-	-	1	0	1	1	1	1
10,3,1 v_3	-	-	-	1	1	1	1	1	1
Step 2:									
Register r	-	-	-	5	2	4	6	8	7
Update rank	4	2	1	6	3	5	7	9	8
Update s	00	00	00	0	0	0	-	-	-
	10	00	00	0	0	0	-	-	-
	00	00	00	0	0	0	-	-	-
Step 3:									
Comp. with R=5						Match			
Output						4			

Figure 6: Sequence of operations for 2-D non-recursive median filtering when the window size is (3×3)

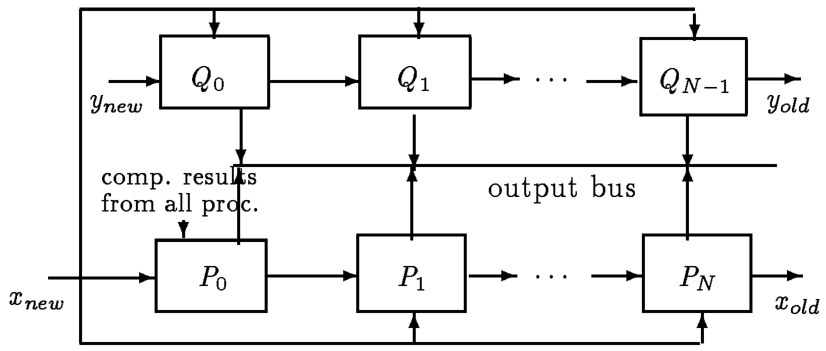


Figure 7: Array architecture for 1-D recursive median filtering without pipelining

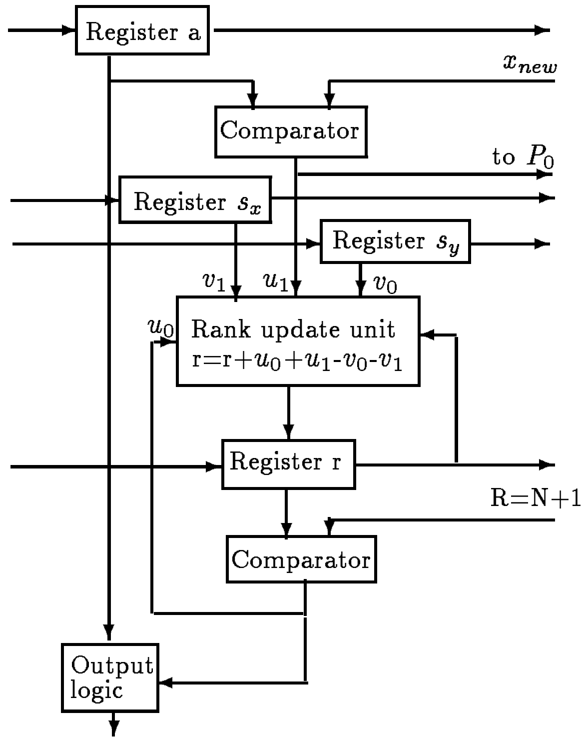


Figure 8: Block diagram of a single processor for 1-D recursive median filtering without pipelining

W_{i-1}={9,9,10,12,2,5}
W_i={9,9,12,2,5,6} x_{new}=6, y_{new}=9, x_{old}=10, y_{old}=9

	Q_0	Q_1	P_0	P_1	P_2	P_3
Step 1:						
Register a	9	9	6	5	2	12
Register s_y	00	00	-	00	00	11
Register s_x	1100	1100	-	0100	0000	1101
Comp. with y_{new}, u_0	0	1	-	0	0	1
Comp. with x_{new}, u_1	1	1	-	0	0	1
Comp. with y_{old}, v_0	0	0	-	0	0	1
Comp. with x_{old}, v_1	0	0	-	0	0	1
Step 2:						
Register r	3	3	-	2	1	6
Update rank	4	5	3	2	1	6
Update s_y	00	10	00	00	00	11
Update s_x	1110	1110	0110	0010	0000	1110
Step 3:						
Compare with ranks 3 and 4	Match	g=1	Match f=0			e=1
		fg+ef+eg=1 ⇒ median is rank 3				
Output median			6			
Compute u_0	1	1	0	0	0	1

Figure 9: Sequence of operations for 1-D recursive median filtering pipelined to 2 levels