NOVEL SORTING NETWORK-BASED ARCHITECTURES FOR RANK ORDER FILTERS

Chaitali Chakrabarti and Li-Yu Wang
Dept. of Electrical Engineering and Telecommunications Research Center
Arizona State University, Tempe, AZ 85287-5706, USA
chaitali@asu.edu

ABSTRACT

This paper presents two novel sorting network-based architectures for computing high sample rate non recursive rank order filters. The proposed architectures consist of significantly fewer comparators than existing architectures that are based on bubble-sort and Batcher's odd-even merge sort. The reduction in the number of comparators is obtained by sorting the columns of the window only once, and by merging the sorted columns in a way such that the set of candidate elements for the output is very small. The number of comparators per output are reduced even further by processing a block of outputs at a time.

1. INTRODUCTION

Rank order filters are non-linear digital filters that perform well in situations in which linear filters fail. In rank order filters, the center value of each window is replaced by the rth largest element in the window, referred to as the element with rank r. Let $W_{i,j}$ be a $(K \times K)$ window centered at (i,j). Then the (i,j)th output, $y_{i,j}$, is the element with rank r in $W_{i,j}$. Rank order filters have many important properties, including reduction of high frequency and impulsive noises without destruction of edge information. These properties make rank order filters a popular pre-processor in many image processing and video processing applications.

The existing architectures for 2-dimensional rank order filters can be classified into array-based architectures, stack filter-based architectures and sorting network based architectures. Of these, only the sorting network-based architectures can be pipelined to any level without significant increase in the area. In the past these networks were either based on bubble-sort [3, 5] and Batcher's odd-even merge sort [2], and required large number of comparators, or were based on running merge sort [6] and required large intermediate storage.

In this paper we propose two novel sorting network based architectures for two-dimensional non-recursive non-separable rank order filters. The proposed architectures require significantly fewer comparators than existing sorting network-based architectures [2, 3, 5]. In both the proposed algorithms, the columns of the window are sorted only once, and the sorted columns are merged in a way such that the set of candidate elements for the output is considerably smaller than the size of the window. The small size of the set of candidate elements makes the number of comparators in these networks significantly small. A further reduction in the number of comparators per output is achieved by processing a block of outputs at a time. In such a scheme, the chip area increases, but since the comparators can be operated at reduced supply voltages, the overall power dissipation reduces.

This work was supported in part by a grant from NSF MIP-9309504.

2. EXISTING ARCHITECTURES

The existing sorting networks for rank order filters are based on bubble-sort [3, 5], on Batcher's odd-even merge sort [2], and on running order sort [6], [4]. The general-purpose bubble-sort network for sorting $M = K^2$ elements consists of M stages with $\lfloor M/2 \rfloor$ comparators per stage. Though the interconnection between stages is small, the number of comparators is $O(M^2)$. Batcher's odd-even merge network, on the other hand, consists of only $O(M \log^2 M)$ comparators but has more complex interconnection between stages. Pitas's running order merge sorter [6] reduces the number of comparisons drastically to $2(M-1) - \log M$, at the expense of a very large increase in intermediate storage as well as in latency. While for line scan mode, all 2-D filters require a storage of $O(\sqrt{M}N)$, where N is the number of pixels per row, Pitas's sorter requires a storage of $O(M^{3/2}N)$. The large storage and the large latency of this serial sorter makes direct implementation of this algorithm unattractive.

Sorting networks for rank order filters are derived from general-purpose sorting networks by eliminating the comparators which do not contribute to the computation of that particular rank. All these architectures can be pipelined to the bit level for most significant bit first implementation by the method of [3].

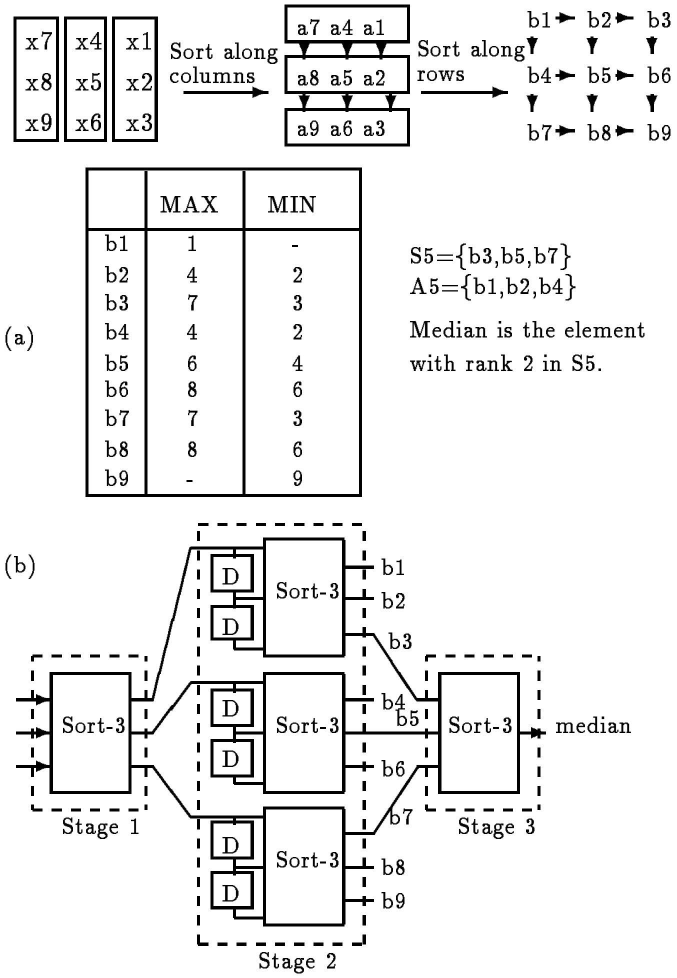

Figure 1: Computation of the median of a (3×3) window. (a) Steps describing Algorithm 1 (b) Network 1

Rank r	Network1	Network2	Batcher's odd -even merge
1	5	5	9
2	9	11	20
3	13	13	20
4	16	15	22
5	13	15	22
6	16	15	22
7	13	13	20
8	9	11	20
9	5	5	9

Figure 2: Comparison of the number of comparators for finding the element with rank r in a (3×3) window.

3. PROPOSED SORTING NETWORKS

In this section we describe two sorting networks which require significantly fewer comparators than the networks of [5, 2]. Both the networks sort the columns of the window first, and then merge the sorted columns. The two networks differ in the way the sorted columns are merged. We assume that the computation window is of size $(K \times K)$, and that it moves across the image from left to right, and from top to bottom. We define O_r to be the element with rank r in the $(K \times K)$ window.

3.1. NETWORK 1

Network 1 implements the following algorithm to find O_r.

Algorithm 1:
Step 1. Sort the 2-D window along the columns.
Step 2. Sort the sorted columns along the rows.
Step 3. Form S_r, a set of candidate elements for O_r, and A_r, a set of elements which are larger than S_r. O_r is the element with rank $r - |A_r|$ in S_r.

This algorithm can be mapped into a sorting network consisting of 3 stages, where stages 1, 2, and 3 implement steps 1, 2, and 3 of the proposed algorithm. Batcher's odd-even merge is used to sort the elements in Steps 1, 2, and 3 (see Fig.1).

We next prove the correctness of the proposed algorithm. After Step 2 of Algorithm 1, a 2-dimensional dependency is established between the K^2 elements of the window. This dependency is used to select S_r, the set of candidate elements for O_r. The procedure for selecting candidate elements is as follows. The rank of each element is bounded by a maximum rank (MAX) and a minimum rank (MIN). The maximum and minimum ranks of the elements can be easily determined from the dependencies between the elements. If the maximum rank and the minimum rank of an element satisfy the condition $MAX \geq r \geq MIN$, then that element belongs to the set S_r. Let A_r be the set of elements which have $MAX < r$. Then the elements in A_r are larger than the elements in S_r, and O_r is the element with rank $\hat{r} = r - |A_r|$ in S_r. Note that S_r and A_r are different for different rank order filters.

We explain the above algorithm by describing the steps in the computation of the median of a (3×3) window. In Step 1, the columns are sorted as shown in Figure 1a. A 2-dimensional dependency graph is obtained after sorting along the rows in Step 2. Since only b_3, b_5, and b_7, satisfy the condition $MAX \geq 5 \geq MIN$, $S_5 = \{b_3, b_5, b_7\}$. Since $A_5 = \{b_1, b_2, b_4\}$, Step 3 of the algorithm computes the element with rank $5 - 3 = 2$ in S_5. Figure 1b describes Network 1 for median computation of a (3×3) window. The number of comparators

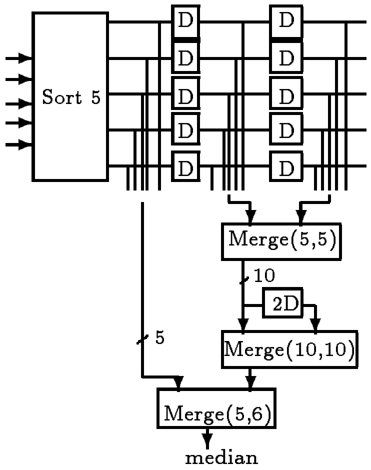

Figure 3: Network 2 for computing the median of a (5×5) window.

is only 13, compared to 22 in [2]. Figure 2 lists the number of comparators for finding the element with rank r in a (3×3) window, $1 \leq r \leq 9$.

The number of comparators in Stage 2 can be reduced by observing that two consecutive windows share $(K-1)$ columns, and so it is not necessary to sort all K elements along the rows every time. Instead, the sorted list of K elements is updated every sample period by removing the oldest element and inserting the newest element. The modified Stage 2 unit then consists of only K comparators for sorting along the row.

3.2. NETWORK 2

We next propose another sorting algorithm which is based on creating sorted lists of size K and $K(K-1)$, and then merging only a *small* subset of the sorted elements. This is in contrast to Batcher's odd-even merge sort, where in the last stage sorted lists of size $K\lceil K/2\rceil$ and $K\lfloor K/2\rfloor$ are merged.

Algorithm 2:
Step 1. Sort $K(K-1)$ elements by merging $(K-1)$ sorted columns.
Step 2. Form S_r, a set of candidate elements for O_r, and A_r, a set of elements which are larger than S_r from the sorted sequences of sizes $K(K-1)$ and K. O_r is the element with rank $r - |A_r|$ in S_r.

In Step 1 of Algorithm 2, the $(K-1)$ sorted columns are merged recursively by a procedure similar to that of [6]. The sorted columns are however merged in parallel.

In Step 2 of Algorithm 2, only a subset of the K^2 elements is used to determine the element with rank r. The procedure is similar to that of Algorithm 1. Let SEQ1 and SEQ2 be the two sorted sequences of sizes $K(K-1)$ and K respectively. Since SEQ1 and SEQ2 are sorted, the maximum and minimum ranks that elements in SEQ1 and SEQ2 can have, can be determined. For instance, the pth largest element of SEQ1 will have minimum rank of p and a maximum rank of $K+p$. All the elements in SEQ1 and SEQ2 which satisfy the condition $MAX \geq r \geq MIN$ belong to the set S_r, and all the elements which satisfy the condition $MAX < r$ belong to the set A_r. Since the elements in A_r are larger than the elements in S_r, O_r is the element with rank $\hat{r} = r - |A_r|$ in S_r. For median computation, S_r consists of the middle $K+1$ elements of SEQ1 and all K elements of SEQ2, A_r consists of $(\lceil K^2/2\rceil - K - 1)$ elements, and O_r is the element with rank $K+1$ in S_r. Figure 3 shows the median network for a (5×5) window. The last merge unit in the median network now finds the median of $5+6=11$ elements, instead of 25. The number of comparators for this network is only 70, compared to 113 if the median network based on Batcher's odd-even merge is used.

4. BLOCK PROCESSING

The number of comparators per output can be reduced even further by processing multiple outputs at the same time, and by presorting the common elements among consecutive windows. In an earlier work [2], we have developed procedures to process a block of outputs for computing 1-D and 2-D median filters. The procedure for processing 1-D windows of size K and block size B consists of (a) recursively sorting disjoint groups of $\lceil K/2\rceil$ elements which occur with the maximum frequency, (b) recursively sorting the remaining $\lfloor K/2\rfloor$ elements of each window, and (c) merging the sorted sequences obtained in steps (a) and (b). For rank order filtering, the merge unit in step (c) is modified appropriately. In this section we apply modified versions of this block processing procedure to the sorting networks described in Section 3. Recently a systematic method for applying block processing to rank order filters has been proposed in [4].

4.1. NETWORK 1

The network for processing B outputs consists of 3 stages; the first stage sorts $(B+K-1)$ columns, the second stage sorts B groups of K elements along each row, and the third stage computes the B outputs. The elements of the second stage are used to form B candidate sets. The third stage then operates on these candidate sets and computes B outputs. The number of comparators in the second stage can be reduced by applying block processing techniques of [2].

4.2. NETWORK 2

In this section, we first describe a procedure for block processing 1-dimensional windows, and then show how this method can be applied to Network 2 for block processing 2-dimensional windows.

(a)

K=9, B=3: y1=median{x1,x2,x3,x4,x5,x6,x7,x8,x9}
y2=median{x2,x3,x4,x5,x6,x7,x8,x9,x10}
y3=median{x3,x4,x5,x6,x7,x8,x9,x10,x11}

Step 1: G0={(x2,x3),(x4,x5),(x6,x7),(x8,x9)}={A,B,C,D},
G1={(x4,x5),(x6,x7),(x8,x9),(x10,x11)}={B,C,D,E}

Step 2: Merge (i) B and C (ii) A and D (iii) D and E.
Merge (B,C) with (A,D) to get G0, and merge (B,C) with (D,E) to get G1.

Step 3: Merge elements with ranks 4 and 5 of G0 (i) with x1 to get y1, and (ii) with x10 to get y2. Merge elements with ranks 4 and 5 of G1 with x3 to get y3.

(b)

Sorting network:

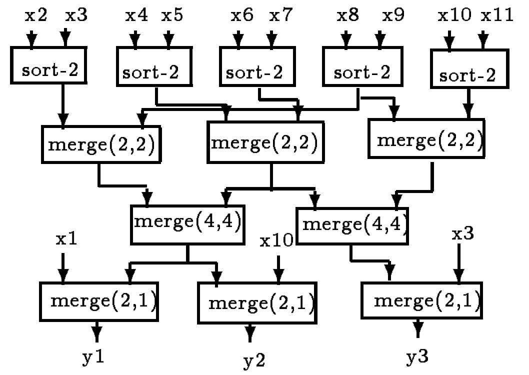

Figure 4: Network 2 for median computation of 1-D windows with $K = 9$ and $B = 3$

The algorithm for block processing of 1-dimensional windows is summarized as follows.

Step 1. Identify the $K-1$ common elements between consecutive outputs y_{2m} and y_{2m+1}, and group them in G_m, $m = 0, 1, 2, \ldots, \lfloor B/2 \rfloor$. Pair the $(2i)$th and the $(2i+1)$th elements in the group to form the ith subgroup, $0 \leq i < (K-1)/2$.

Step 2. Apply the block processing procedure of [2] to $\lceil B/2 \rceil$ sets of $(K-1)/2 = M$ subgroups. This involves (a) recursively sorting $\lceil M/2 \rceil$ subgroups which occur with the maximum frequency, (b) recursively sorting the remaining $\lfloor M/2 \rfloor$ subgroups in each group, and (c) merging the sorted sequences obtained in steps (a) and (b). The merge units in step (c) merges sequences of sizes $2\lceil M/2 \rceil$ and $2\lfloor M/2 \rfloor$, and outputs only 2 elements, namely, the elements with ranks r and $r-1$.

Step 3. The two elements from the merge unit in Step 2 are used to compute 2 outputs. Let W_{2m} and W_{2m+1} be the windows corresponding to outputs y_{2m} and y_{2m+1}. Then the two elements obtained from the mth merge unit in Step 2 are merged with $\{W_{2m} - G_m\}$ and $\{W_{2m+1} - G_m\}$ to compute y_{2m} and y_{2m+1} respectively. The output is the median of the 3 elements.

Figure 4 describes an example for median computation with $K = 9$ and $B = 3$. The number of comparators of this network is only 34, compared to 49 in our previous realization in [2].

Number of comparators/output for a (5x5) window, r=13		
B	Network 1	Network 2
1	80	57
2	58	33.5
3	59.3	42.7
4	61.5	31.3
5	57.8	37.2
6	57.8	30.5

Figure 5: Comparison of number of comparators per output for median computation of (5×5) windows for different block sizes.

The algorithm for 1-D block processing can be easily extended to 2-D by replacing each element of the 1-D window by a column (or row) of K sorted elements. The merge units in Step 2c of the algorithm now merge sorted sequences of sizes $K\lceil (K-1)/2 \rceil$ and $K\lfloor (K-1)/2 \rfloor$. A set of candidate elements are selected from the two sorted sequences of sizes $K(K-1)/2$ and K, and the merge unit in Step 3 operates on this candidate set. The procedure to select the elements of the candidate set is very similar to that of Algorithm 2 (in 3.2). For instance, for median computation, the merge unit in Step 3 merges $K+1$ elements with K elements, and finds the median of the $(2K+1)$ elements. Figure 5 lists the number of comparators per output for different block sizes for a (5×5) window. Note that for all B, the number of comparators per output for Network 2 is significantly less than that of Network 1.

5. REFERENCES

[1] K. E. Batcher, "Sorting network and their applications," 1968 Spring Joint Computer Conference, AFIPS Proc. vol. 32, pp. 307-314.

[2] C. Chakrabarti "Sorting Network-based Architectures for Median Filters," IEEE Trans. on Circuits and Systems, pp.723-727, Nov 1993.

[3] M. Karaman, L. Onural, A. Atalar, "Design and implementation of a general purpose median filter in CMOS VLSI," IEEE Journal of Solid State Circuits, vol. 25, April 1990.

[4] L. Lucke, K. Parhi, "Parallel Processing Architectures for Rank Order and Stack Filters," IEEE Trans on Signal Processing, May 1994.

[5] K. Oflazer, "Design and implementation of a single chip 1-D median filter," IEEE Trans on Acoust., Speech, and Signal Processing, vol. ASSP-30, 1983, pp. 1164-1168.

[6] I. Pitas, "Fast algorithms for running order and max/min calculations," IEEE Trans on Circuits and Systems, vol. 36, June 89, pp. 795-804.