Novel Sorting Network-Based Architectures for Rank Order Filters

Chaitali Chakrabarti and Li-Yu Wang[1]

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, AZ 85287-5706
Ph: (602) 965-9516
Fax: (602) 965-8325
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 sorting network-based 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 number of candidate elements for the output is very small. The number of comparators per output is reduced even further by processing a block of outputs at a time. Block processing procedures that exploit the computational overlap between consecutive windows are developed for both the proposed networks.

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

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. Median filters are rank order filters with $r = \lceil |W|/2 \rceil$, where $|W|$ is the size of the window. These 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.

There are two types of rank order filters: recursive and non-recursive. In recursive rank order filters, the window consists of the recent median values as well as the sampled values of the image, while in non-recursive filters, the window consists of only the sampled values of the image. In this paper we develop sorting network-based architectures for 2-dimensional non-recursive non-separable rank order filters. Let $W_{i,j}$ be a $(K \times K)$ window centered in (i, j). Then $y_{i,j}$ is the element with rank r in $W_{i,j}$. We assume that the window moves across the image from left to right, and from top to bottom, and that an output is computed every sample period.

There are three classes of architectures for 2-dimensional rank order filters: array-based architectures, stack filter-based architectures and sorting network-based architectures. [12] gives an excellent survey of these architectures. The existing array architectures consist of an array of K^2 processors, and either compute an output every K sample periods by having 3 comparators in each processor [6, 8], or compute an output every sample period by having $K+1$ comparators in each processor [3]. If an output is to be computed every sample period, then the large area of the array architecture makes such a design impractical. Bit-parallel implementations of stack filter-based architectures also have large area and suffer from similar drawbacks. A b-bit parallel stack filter for 2-dimensional filtering consists of K threshold decomposition units and $2^b - 1$ bit-level logic (BLL)[2] units. The bit-serial implementation, on the other hand, is very economical in area and requires only 1 BLL unit [5], or b BLL units in the unfolded version [13]. The bit-serial implementations compute the output bits recursively most significant bit first. The sample period of all these implementations is bound by the delay to compute an output bit (which is equivalent to the delay in

[2]The BLL unit can be built with AND gates and OR gates as in a Positive Boolean Function unit, or by a tree of adders and a comparator.

the BLL unit).

The sorting network-based architectures can be pipelined to any level, and consequently their sample periods can be reduced arbitrarily. In the past, these networks were either based on bubble-sort [7, 10] and Batcher's odd-even merge sort [4], and required large number of comparators, or were based on running merge sort [11] and required large intermediate storage. In this work, we develop networks with reduced number of comparators and storage units, making single chip implementations of even moderate-sized windows feasible.

In this paper, we propose two novel sorting network based architectures for rank order filters which require significantly fewer comparators than existing networks [7, 10, 4]. For instance, only 15 comparators are required to find the median of a (3×3) window, instead of 22 comparators if Batcher's odd-even merge sort was used. Though the number of comparators in the proposed networks is more than in [11], both the networks have significantly less storage requirements, making the overall area requirements less than in [11]. 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 number of candidate elements for the output is considerably smaller than the size of the window. The small number of candidate elements makes the number of comparators in the corresponding 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 power dissipation reduces [3].

The rest of the paper is organized as follows. In Section 2, we give a brief description of the existing sorting network architectures. In Section 3, we describe the two proposed sorting algorithms and architectures for rank order filters. Then, in Section 4, we describe ways to process blocks of outputs in the two proposed sorting networks.

2 Existing network architectures

The existing sorting networks for rank order filters are based on bubble-sort [7, 10], on Batcher's odd-even merge sort [4], and on running order sort [11]. 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

[3] Power dissipation is proportional to the square of the supply voltage.

merge network, on the other hand, consists of only $O(M \log^2 M)$ comparators, but has more complex interconnection between stages. The odd-even merge network parallely sorts two subsequences of sizes M_1 and $M - M_1$, and then merges the two sorted subsequences. The network is most efficient when $M_1 \approx M/2$. Pitas's running order merge sorter [11] reduces the number of comparisons drastically to $2(M-1) - \log M$, at the expense of large increases in the intermediate storage and 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. There are other general-purpose 2-D sorting algorithms such as shear sort and bitonic sort, where the rows and columns are sorted alternately in $\log M + 1$ passes. These algorithms do not exploit the computational redundancies of running order computations, and are thus less suitable for moving window applications.

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 the desired rank. For example, to compute the median of 9 elements, 4 comparators can be eliminated from the last merge unit of the corresponding Batcher's odd-even merge sorting network.

The sorting network-based architectures based on bubble-sort and odd-even merge sort are non-recursive and can be pipelined to any level by placing latches in the feed-forward paths. In fact, the existing architectures as well as the proposed architectures can be pipelined to the bit level for most significant bit first implementation by the method of [7].

3 Proposed sorting networks

In this section we describe two sorting networks which require significantly fewer comparators than the networks of [10, 4]. Both the proposed 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. However, both the networks reduce the number of comparators by reducing the number of candidate elements during the merge operations.

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. In every sample period, K new elements along a column are input. We define the output 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:

1. Sort the 2-D window along the columns.
2. Sort the sorted columns along the rows.
3. Find the maximum rank, MAX, and the minimum rank, MIN, for each element. Form (i) the set S_r with elements that satisfy the condition $MAX \geq r \geq MIN$, and (ii) the set A_r with elements that satisfy $MAX < r$. Then O_r is the element with rank $\acute{r} = r - |A_r|$ 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 (see Figure 1). Batcher's odd-even merge is used to sort the elements in Stages 1, 2, and 3.

We next prove the correctness of the proposed algorithm. In 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. Since the dependencies between the elements is known, the maximum possible rank (MAX) and the minimum possible rank (MIN) can be determined for each element. An element is a candidate for the output if r lies between its maximum and minimum possible rank. In other words, if an element satisfies the condition $MAX \geq r \geq MIN$, then that element belongs to the candidate set S_r. Let A_r be the set of elements which are larger than the largest element in S_r. Then an element which satisfies the condition $MAX < r$ belongs to the set A_r. Thus O_r is the element with rank $\acute{r} = r - |A_r|$ 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 2a. A 2-dimensional dependency graph is obtained after sorting along the rows in Step 2. Since only $b3$, $b5$, and $b7$, satisfy the condition $MAX \geq 5 \geq MIN$, $S_5 = \{b3, b5, b7\}$. Since $A_5 = \{b1, b2, b4\}$, Step 3 of the algorithm computes O_5, the element with rank $5-3=2$ in S_5. Figure 2b describes Network 1 for median computation of a (3×3) window. The number of comparators is only 13, compared to 22 in [4]. Table 1 lists the number of comparators for finding O_r, $1\le r\le 9$, for (3×3) windows.

For (5×5) windows, the median is the element with rank 13. Here the number of candidate elements is 13, and there are 6 elements which are larger than the candidate elements. Thus the output is the element with rank $13-6=7$ among the candidate elements. The number of comparators in the corresponding network is only 80, compared to 113 if Batcher's odd-even merge sort is used. Table 2 lists the number of comparators for finding O_r, $1\le r\le 25$, for (5×5) windows.

Modifications in Stage 2 of Network 1:

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 by removing the oldest element and inserting the newest element. The sorted list is updated every sample period by a modified version of the method proposed in [1]. The modified Stage 2 unit 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 two sorted lists one of size K and the other of size $K(K-1)$, and then merging only a *small* subset of the elements in the sorted lists. In contrast, the last stage of Batcher's odd-even merge sort merges sorted lists of size $K\lceil K/2\rceil$ and $K\lfloor K/2\rfloor$.

Algorithm 2:

1. Sort the 2-D window along the columns.
2. Merge $K-1$ sorted columns to get a sorted list of size $K(K-1)$.
3. Find the maximum rank, MAX, and the minimum rank, MIN, for each element. Form (i) the candidate set S_r with elements that satisfy $MAX\ge r\le MIN$, and (ii) the set A_r with elements that satisfy $MAX<r$. Then O_r is the element with rank $\acute{r}=r-|A_r|$ in S_r.

In Step 2 of Algorithm 2, the $(K-1)$ sorted columns are merged recursively by a procedure similar the one in [11]. The main difference is that here the sorted elements are available in parallel, and that each merge operation is based on Batcher's odd-even merge sort.

In Step 3 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 of elements in SEQ1 and SEQ2 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 $\acute{r} = r - |A_r|$ 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 elements in Batcher's odd-even merge network. The number of comparators for this network is only 57, compared to 113 in Batcher's odd-even merge network.

3.3 Comparisons

Tables 1 and 2 compare the number of comparators required by Networks 1, 2, and Batcher's odd-even merge for finding the element with rank r. Notice that both Networks 1 and 2 require significantly fewer comparators than Batcher's odd-even merge network. Moreover, Network 2 requires fewer comparators than Network 1 for most ranks. Since the merge operations of all the networks are done by Batcher's odd-even merge, the complexity of interconnection is proportional to the number of comparators. Consequently, Network 2 has the least interconnection complexity, and would occupy smaller area. All three networks require a storage of KN for line-scan mode processing, where N is the number of pixels per row. Networks 1 and 2 require additional intermediate storage. Specifically, Network 1 requires an intermediate storage of size $K(K-1)$, while Network 2 requires an intermediate storage of size $K(K^2-2K+3)/3$. All three networks can be pipelined to the bit level for most significant bit first implementation [7].

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 [4], we have developed block processing procedures for 1-D and 2-D median filters. The procedure for processing 1-D windows of size K and block size B is summarized below [4]:

1. Recursively sort disjoint groups of $\lceil K/2 \rceil$ elements which occur with the maximum frequency.

2. Recursively sort the remaining $\lfloor K/2 \rfloor$ elements of each window.

3. Merge the sorted sequences obtained in steps 1 and 2.

For rank order filtering, the merge unit in step 3 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 [9]. The method uses greedy heursitics to identify input sets which are common to maximum number of outputs.

4.1 Network 1

When B outputs are processed at the same time using Algorithm 1, there is overlap in the computations in Step 1 and in Step 2. This computational overlap is mapped into sharing of comparators in Stages 1 and 2 of the corresponding network.

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. These candidate sets are same as those of Algorithm 1. The third stage operates on the B candidate sets and computes B outputs. The number of comparators in the second stage can be reduced by applying block processing techniques of [4]. For example, for $B = 2$ and $K = 3$, the number of comparators in the second stage can be reduced from $2 \times 7 = 14$ to 11 by applying such techniques. Figure 4 illustrates this procedure. Tables 3a and 3b list the number of comparators per output for different block sizes for (3×3) and (5×5) windows respectively.

4.2 Network 2

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

The algorithm for block processing of 1-D windows of size K and block size B is motivated by the following observations: (i) Since two consecutive windows share $K-1$ elements, the number of comparisons per output can be reduced by sorting the $K-1$ common elements first and then merging with the remaining element. (ii) For rank order filters, it is not necessary to completely sort the $K-1$ common elements. Instead, it is sufficient to generate only the elements with ranks r and $r-1$. (iii) The $K-1$ common elements can be sorted efficiently by merging sorted lists of approximately equal size in each stage of Batcher's odd-even merge network. The algorithm is described below.

Algorithm 3:

1. Identify the $K-1$ common elements among consecutive windows W_{2m} and W_{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 super-element, $0 \leq i < (K-1)/2$. At the end of this step there are $\lceil B/2 \rceil$ groups and $(K-1)/2 = M$ super-elements per subgroup.

2. Apply the method in [4] to $\lceil B/2 \rceil$ groups:

   (a) Recursively sort $\lceil M/2 \rceil$ super-elements which occur with the maximum frequency

   (b) Recursively sort the remaining $\lfloor M/2 \rfloor$ super-elements in each group

   (c) Merge the sorted sequences obtained in steps a and b.

   The merge units in step c merge sequences of sizes $2\lceil M/2 \rceil$ and $2\lfloor M/2 \rfloor$, and outputs the elements with ranks r and $r-1$.

3. The two elements obtained from the mth merge unit in Step 2c are merged with $\{W_{2m} - G_m\}$ and $\{W_{2m+1} - G_m\}$ to compute the outputs y_{2m} and y_{2m+1} respectively. In each case, the output is the median of the three elements.

Figure 5 describes an example for median computation with $K = 9$ and $B = 3$. Each of the groups $G0$ and $G1$ contain 8 elements which are further grouped into 4 super-elements. The block processing procedure of [4] is then applied to the 2 groups. The number of comparators of this network is only 34, compared to 49 in our previous realization in [4], and 60 in [8]. Table 4 compares the number of comparators used by this method with those of [4], and [8] for different values of K and B.

In Algorithm 3, if the block size B is even, then the number of comparisons in the implementation of Step 1 can be further reduced by exploiting the fact that all input pairs of the form $(x_{j+\alpha B}, x_{j+\alpha B+1})$, where α is a positive integer, can be sorted by a single comparator. This comes at the expense of additional memory which is used to store the values of the previous comparisons.

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. 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 6 describes the sorting network for median computation of (5×5) windows when $B = 4$. Note that one of the sorted sequences of size 10 that is input to the second merge(10,10) unit is obtained by delaying a sorted list of size 10 by 4 time units.

Implementation:

A (3×3) median filter with $B = 2$ has been implemented using Berkeley CAD Tools. The technology is 2 micron n-well CMOS. The architecture consists of one sort-3 unit, one merge(3,3) unit, two merge(3,4) units, nine 8-bit shift registers, six 3-bit shift registers and latches. The total die size (excluding the I/O pads and the line memory) is 6 sq. mm. The delay in the compare-swap unit is less than 8 ns This makes the bit-level pipelined implementation of this network support very high sample rates.

4.3 Comparisons

Table 3 compares the number of comparators per output for different block sizes for Networks 1 and 2. For all B, the number of comparators per output for Network 2 is significantly less than that of Network 1. This is because computational overlap cannot be exploited efficiently in Network 1. Recall that there are B Stage 3 units in the third stage. So, even though, computational overlap is used to reduce the number of comparators per output in Stage 1 and Stage 2, the overall reduction in the number of comparators per output is not that significant in Network 1.

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, pp. 172-181, 1984.

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

[3] C. Chakrabarti, "High Sample Rate Array Architectures for Median Filters," IEEE Trans on Signal Processing, pp. 707-712, March 1994.

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

[5] K. Chen, "Bit-serial realizations of a class of non-linear filters based on positive Boolean functions," IEEE Trans on Circuits and Systems, pp. 785-794, June 1989.

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

[7] 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.

[8] L. Lucke, K. Parhi, "Parallel Structures for Rank Order and Stack Filters," Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 1992.

[9] L. Lucke, K. Parhi, "Parallel Processing Architectures for Rank Order and Stack Filters," IEEE Trans on Signal Processing, pp. 1178-1189, May 1994.

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

Table 1: Comparison of the number of comparators for the finding the element with rank r in a (3×3) window.

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

[11] I. Pitas, "Fast algorithms for running order and max/min calculations," IEEE Trans on Circuits and Systems, vol. 36, pp. 795-804, June 1989.

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

[13] P. Ruetz, "The Architectures and Design of a 20MHz Real-Time DSP Chip Set," IEEE Journal of Solid-State Circuits, vol. 24, pp. 338-348, April 1989.

Rank r	Network1	Network2	Batcher's odd-even merge
1	8	9	24
2	21	33	94
3	27	35	94
4	42	45	104
5	49	47	104
6	60	52	108
7	57	52	108
8	66	56	112
9	70	55	112
10	77	58	113
11	73	56	113
12	80	59	113
13	80	57	113
14	80	59	113
15	73	56	113
16	77	58	113
17	70	55	112
18	66	56	112
19	57	52	108
20	60	52	108
21	49	47	104
22	42	45	104
23	27	35	94
24	21	33	94
25	8	9	24

Table 2: Comparison of the number of comparators for the finding the element with rank r in a (5×5) window.

(a)

B	Number of comparators per output for a (3x3) window. r=5.	
	Network1	Network2
1	13	15
2	10	10.5
3	10	11
4	9.2	9.7

(b)

B	Number of comparators per output for a (5x5) window. r=13	
	Network1	Network2
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

Table 3: Comparison of the number of comparators per output for median computation for different block sizes in (a) (3×3) windows and (b) (5×5) windows

Number of comparators				
K	B	Ref [8]	Ref [4]	Proposed method
3	6	15	15	15
5	4	27	24	16
5	5	24	28	23
7	6	65	59	39
9	3	60	48	34

Table 4: Comparison of the number of comparators for various values of K and B.

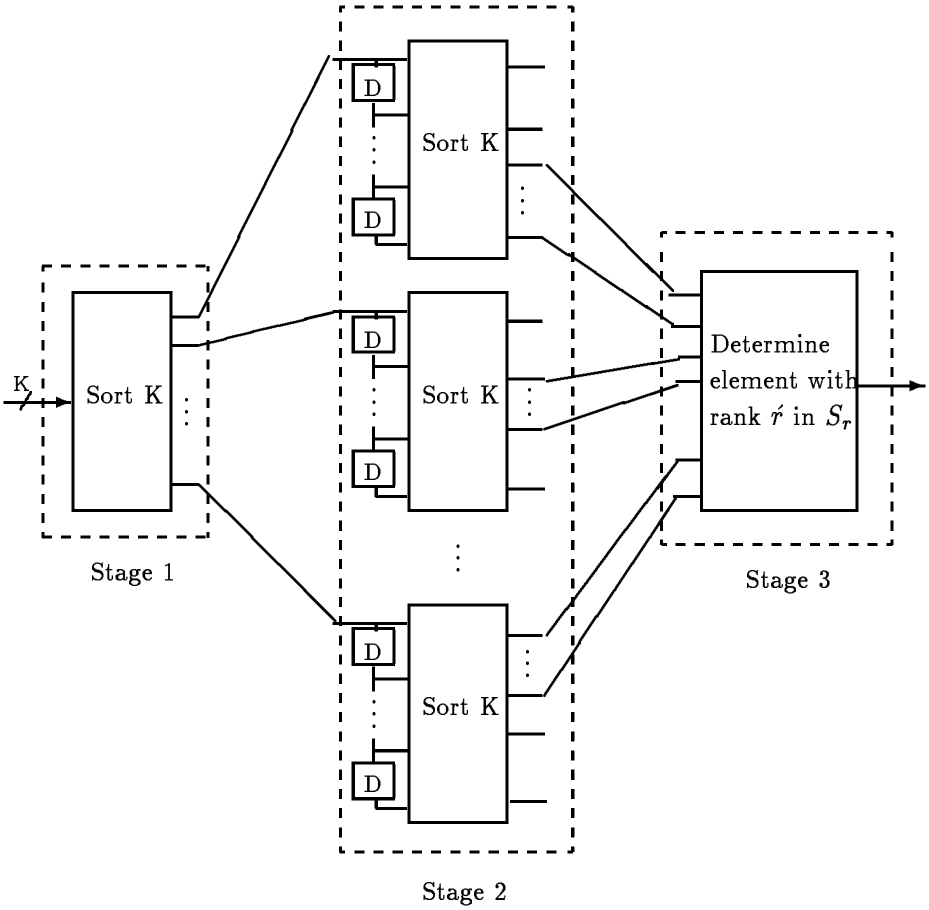
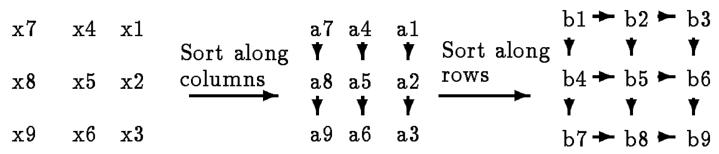


Figure 1: Block diagram of Network 1. Stage 1 sorts K elements along a column, Stage 2 sorts K elements along a row, and Stage 3 determines the element with rank $\acute{r} = r - |A_r|$ in the candidate set S_r, $|A_r|$ is the number of elements larger than S_r.



(a)

	MAX	MIN
b1	1	-
b2	4	2
b3	7	3
b4	4	2
b5	6	4
b6	8	6
b7	7	3
b8	8	6
b9	-	9

S_5={b3,b5,b7}
A_5={b1,b2,b4}

Median is the element with rank 5-3=2 in S_5.

(b)

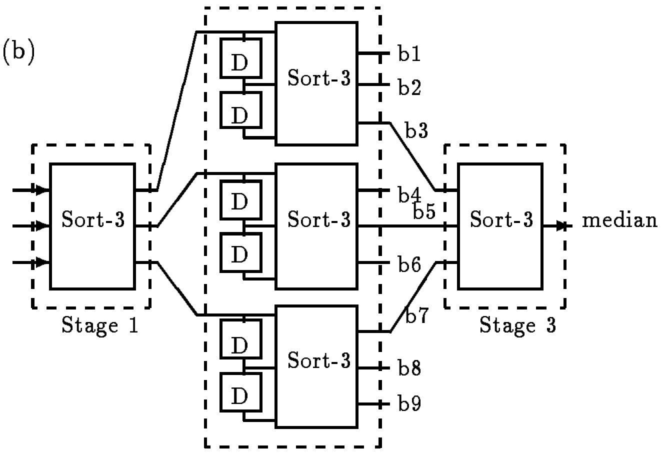


Figure 2: (a) Steps describing the computation of the median of a (3×3) window using Algorithm 1. (b) Network 1 for computing the median of a (3×3) window. Total number of comparators is 3+2+3+2+3=13.

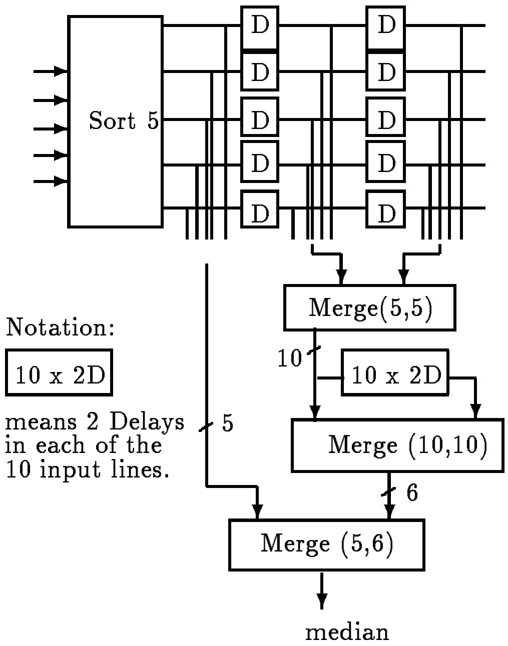
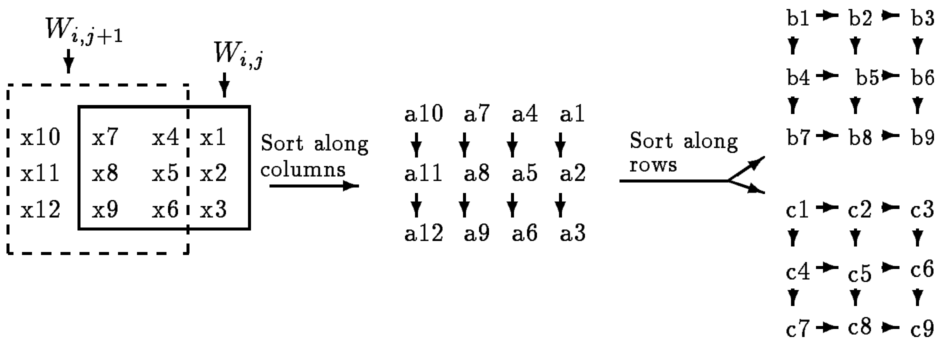
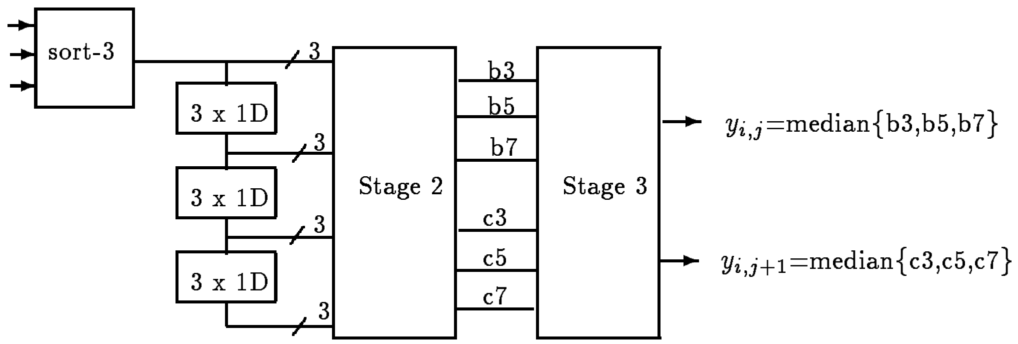


Figure 3: Network 2 for computing the median of a (5×5) window. The number of comparators is 57.



Block diagram of the sorting network:



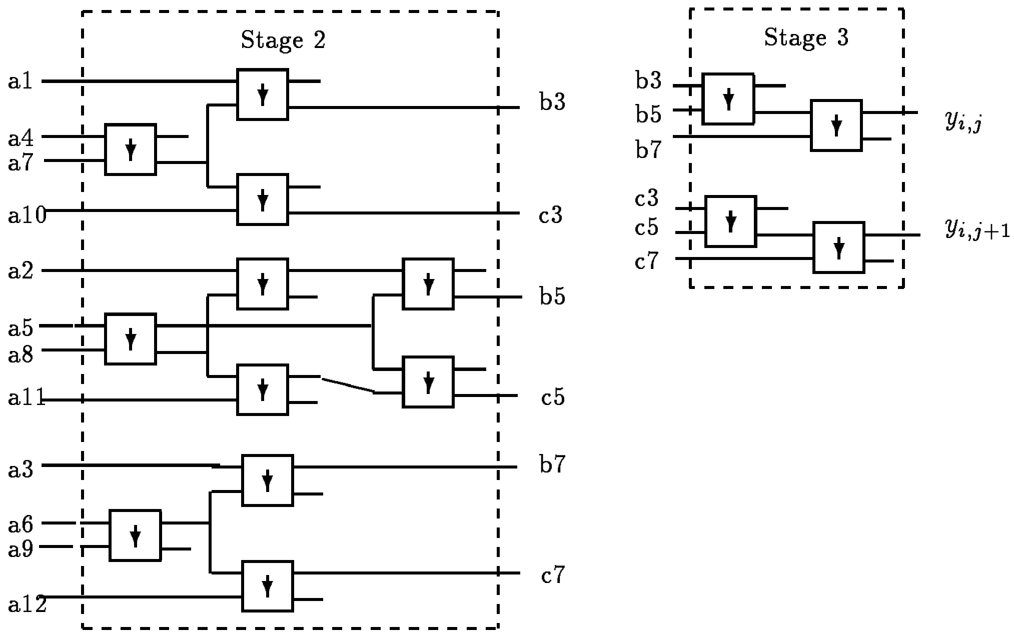


Figure 4: Network 1 for median computation of a (3×3) window with $B = 2$.

(a) y_5=median{x1,x2,x3,x4,x5,x6,x7,x8,x9}
K=9, B=3 y_6=median{x2,x3,x4,x5,x6,x7,x8,x9,x10}
y_7=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 y_5, and (ii) with x10 to get y_6. Merge elements with ranks 4 and 5 of G1 with x3 to get y_7.

(b) **Sorting network**:

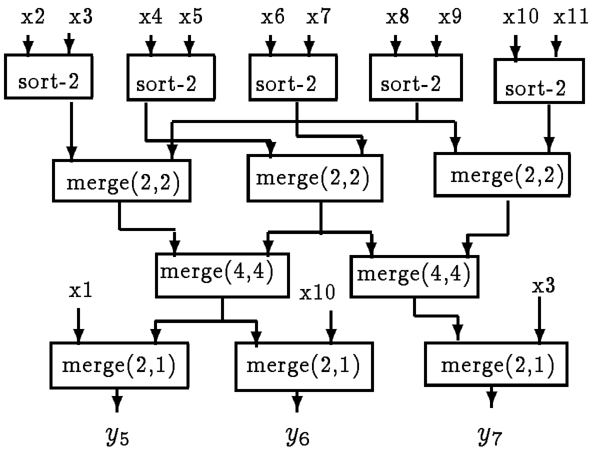


Figure 5: Network 2 for median computation of 1-D windows with $K = 9$ and $B = 3$.

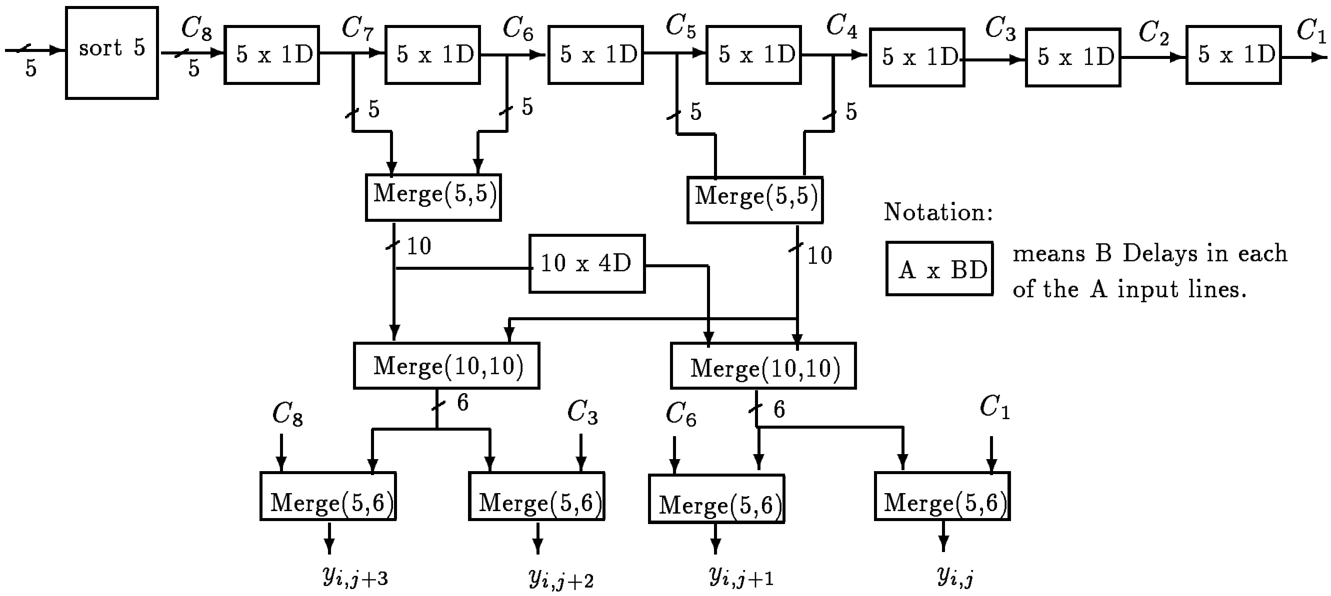


Figure 6: Network 2 for median computation of (5×5) windows with $B = 4$. C_i, $i = 1$ through 8, are the sorted columns.