Understanding A

Readability A

Extent of research A

New approach: A+

A method by which 2 new samples are input every clock period & 2 medians are output " "

14+9+9+10 = 42

42/45

High Sample Rate VLSI Median Filters

Gagan Gupta
992-56-5432
EEE 598D

Abstract

Median filters have been proposed for the analysis of speech data & in image processing to enhance the data by smoothing the signal & removing noise. Many designs for median filters have been suggested in literature. In this paper, we will introduce new high sample rate VLSI median filter designs that can work at a rate faster than the existing designs. Designs for 1 dimensional & 2 dimensional cases are discussed. The same ideas can be easily extended over to any order statistic filters.

I. Introduction

Median filters are a special class of nonlinear filters useful in the removal of impulse noise & smoothing signals. These filters while doing so, preserve the edge information thereby making them attractive for speech processing & image processing applications. Mathematical properties of these filters have been discussed in [1] & [2]. There are two classes of median filters. The 1 dimensional filters are for speech processing & related problems; 2 dimensional filters are often used in image processing. We will discuss 1 dimensional filters & show how the same idea can be extended over for 2 dimensional filters.

1 dimensional filters work as following. A window of size 2N+1 is moved across the entire signal & the center value of each window is replaced by the median values in the window. A window centered on the ith value, of an input stream of values $X_0, X_1, X_2, \ldots,$ is denoted as $W_i = \{X_{i-N}, \ldots, X_i, \ldots, X_{i+N}\}$. The window W slides over the input stream as shown in figure 1.The idea of a median filter is to use the median of the values in the window W_i as the best guess for the true value of X_i. The running median Y_i = median $\{W_i\}$. In general any order statistic could be used in place of the median , i.e. Y_i = rank_r $\{W_i\}$ where the element of rank r is the rth largest in a set. We will describe algorithms for median values. In general the same algorithms can be used for any order statistic values.

The existing architectures for median filters can be broadly classified into two classes [5]: ones which adopt window oriented approach & others which adopt sample oriented approach. In window oriented approach, samples are stored in order of arrival & are either linear systolic array architectures [6] or are based on sorting networks [7] (see figure 2). Although the 2 dimensional sorting networks can be highly pipelined, they use up a lot of space $O(|W|^2)$, $|W| = 2N+1$, making them unattractive for large windows [7]. Architecture of [6] consists of 2N+1 processors, but has a large pipeline period because for each input sample a sequence of operations are performed for updating ranks. The sample oriented approach maintains the sample in sorted order & ranks are updated based on the sample values arriving & departing [5, 9] (see figure 3). They use 2N+1 processors, but have a large pipeline period because for each input sample, comparisons & shifts have to be performed to maintain the sorted list.

There are several proposed architectures for median filters. [3] gives a survey & critical analysis of the existing architectures. There are algorithms that take time O(N) to output a median value for every window.Architecture of [4] has a pipeline period of 2 (if no interleaving is used), but has a latency time O(N). The 2 dimensional architecture of [4] also uses extra hardware to convert parallel data input into serial input thereby losing the advantage of availability of data in parallel.

We describe two architectures for median filters which allow very high data sampling rate with a low constant latency time & a pipeline period of 1 without any interleaving. The first architecture is same as the samples stored in order of arrival algorithm in [8], but with a small modification which speeds up the original algorithm. The modified algorithm has been repeated here. The second algorithm is based on this algorithm.

II. Design I

The architecture consists of linear systolic array of processors. A high sampling rate is achieved by pipelining the computations in each processor. Each processor performs computations related to windows W_i & W_{i-1} simultaneously. This is done by pipelining the computations through two computation units so that while the first computation unit is working on window W_i, the second performs computations related to window W_{i-1}. The architecture can be sampled at a rate twice that of existing systolic array implementations [4,5,6].

The architecture consists of a linear array of 2N+1 processors, each of which consists of two computing units. The first computing unit consists of register a, two comparators, while the second computing unit consists of a rank register R, a comparator, a few 1 bit flags & circuitry for updating the rank.

There are 2N+1 such processors in the array which store samples in the order of arrival. P_1 stores the oldest sample & P_{2N+1} stores the newest sample. The rank register stores the rank of this sample. The array stores one entire window at a time. Ranking of the present window is obtained by a simple modification of the ranking of the old window (rather than reevaluation from scratch). The algorithm consists of comparisons followed by rank updates. The implementation is pipelined in such a way that while the first computing unit performs comparisons on the ith window, the second computing unit updates the rank of the *i-1*th window. We next describe the sequence of operations when Xi+N is stored in register a in P_{2N+1}; X_{i-N-1} is the old sample to be deleted (see figure 4).

The ranks of each sample is stored in rank register using binary number system. Hence maximum size of rank register will be $\log_2(2N+1)$ bits. Say $k= \log_2(2N+1)$. Then rank register will be $r_k.r_{k-1}....r_1$ where $r_j = 1$ or 0, $1 <= j <= k$. r_1 is the least significant bit & r_k is the most significant bit. There will be a set of unique 2N+1 numbers representing the 2N+1 possible ranks & at a given instance, all elements will have exactly one rank out of this set. A parity (either even or odd) can be defined for bit r_j of all the 2N+1 elements of this set, $1 <= j <= k$. This parity will be unique for a given set of 2N+1 numbers. Keeping this in mind we proceed to describe the algorithm.

Operations in the first computing unit

At the beginning of this operation, register a in processor P_j contains $X_{j+i-N-1}$, $1 <= j <= 2N+1$.

Compare with new & oldest samples

Sample X_{i-N-1} is compared with contents of register a in processors P_1 through P_{2N}. A 1 bit flag V_j in processor P_j is set to 1 if $X_{j+i-N-2} > X_{i-N-1}$ for $2 <= j <= 2N+1$. Sample X_{i+N} is compared with contents of processor P_1 through P_{2N} & 1 bit flag U_j in processor P_j is set to 1 if $X_{j+i-N-2} > X_{i+N}$ for $2 <+ j <= 2N+1$.
At the end of this operation, each processor writes the contents of register a into register b. Register a of all processor shift right & the new sample X_{i+N+1} is written into register a of P_{2N+1}.

Operations in the second computing unit

At the beginning of this set of operation, register b contains the elements of the *i-1*th window, rank register R contains the rank of the elements of the *i-2*th window shifted by 1 & flags U & V contain the comparison results of the *i-1*th window.

Update rank

Each processor updates it's rank register, R_j, depending on the flag values U_j & V_j. The following operations take place in processor P_j, $1 <= j <= 2N$.

If $U_j = V_j$ then $R_j = R_j$
If $U_j > V_j$ then $R_j = R_j+1$
If $U_j < V_j$ then $R_j = R_j-1$

The above algorithm computes rank of samples in processors P_1 to P_{2N}. The rank of the element X_{i+N-1} (in P_{2N+1}) is straightforward to compute. Ranks of 2N elements are known & the parity of the 2N+1 possible rank combinations is already known. A parity generator that takes as input r_j of the rank register R of all processors P_i $1 <= i <= 2N$ will give the value of the r_j bit of the rank register R of the element X_{i+N-1} in P_{2N+1}. $\mathrm{Log}_2(2N+1)$ parity generators will be required.

Rank Matching

Any processor whose rank matches rank N+1 (in case of median) outputs the contents of it's register b. This is the i-1th window median. The rank register of all the processors are shifted right. Contents of rank register of processor P_1 is lost.

Operations in the first & second computing units take place simultaneously. In the second computing unit, computation of the rank of element X_{i+N-1} is done in parallel with rank match. If none of the processors contain rank N+1, X_{i+N-1} is output, otherwise processor with rank register N+1 outputs the content of register b.

The update & rank matching algorithms can be easily implemented in combinational logic. Note that the latency time is independent of N. The pipeline period is 1.

III. Design II

We now discuss a even faster 1 dimensional median filter architecture. The speedup in the previous case has been achieved by modifying existing algorithm [6,8] at implementation level. We achieve a speedup of two over the previous algorithm by proposing a new algorithm, keeping the implementation details almost the same. This speedup is achieved by taking as input multiple samples per time unit & outputing as many median values corresponding to the individual windows being processed at that time instance.

If you consider a window size of 2N+1 with elements X_1 to X_{2N+1} with ranks from 1 to 2N+1, then the median of this window will be the element with rank N+1. X_1 is the oldest element & X_{2N+1} is the most recent element. X_{2N+2} will be the new input sample that will be taken & X_1 will be the element that will be pushed out. Consider a window of size 2N+2 with elements X_1 to X_{2N+2}. This window can be considered to contain two successive windows W_i & W_{i+1} of size 2N+1 each. The window W_i consists of samples X_1 to X_{2N+1} & window W_{i+1} consists of samples X_2 to X_{2N+2} (see figure 5). If both the windows can be processed simultaneously i.e. if medians of both the windows can be found at the same time, two windows will be processed at the same time, thereby enabling two new samples X_{2N+3} & X_{2N+4} to be input & two old samples X_1 & X_2 to be discarded. At this time instance the window will contain the next two windows W_{i+2} (samples X_3 to X_{2N+3}) & W_{i+3} (samples X_4 to X_{2N+4}). The window slides over the input stream by jumping two samples at a time (see figure 6). Effectively this will enable two samples to be taken per time unit thereby doubling the sampling rate, thus making this algorithm faster by a factor of two.

Let us consider a window size of 2N+2 as described above. There are 2N+2 samples in this window with ranks 1 to 2N+2. In general the medians of the two windows Wi & W_{i+1} contains in this window will be the elements with ranks N+1 & N+2 respectively where the sample ranks have been decided by combining both the windows. However there are exceptions to this rule. The complete algorithm follows.

We follow the following notation :

$R(X_i)$	: rank of sample X_i where $1 <= i <= 2N+2$
X_1, X_2	: oldest samples
X_{2N+1}, X_{2N+2}	: recent samples
$X_{r=i}$	: sample with rank i $1 <= i <= 2N+2$
M_j	: median of window W_j
R_i	: Rank register value of processor P_i

The array architecture is the same as that of Design I, except that we now have 2N+2 processors. The processors of this architecture are also very similar to the processors of the previous architecture. Each processor has two computing units. In this case the first computing unit requires 4 comparators & a few more one bit flags. The operations in the two computing units are slightly modified. The changes are described below.

The primary change in this design apart form the algorithm is the way the ranks are being stored. Ranks are no longer stored as numbers using the binary number system. For a window size of 2N+2, rank register has 2N+2 bits. Rank register of processor P_i is R_i & can be defined as

$R_i = \{ r_j \; ; \; 1 <= j <= 2N+2 \; ; \; r_j = 0,1\}$ alternatively
$R_i = r_{2N+2}.r_{2N+1}........r_j.......r_2.r_1$ where $r_j = 1$ or 0 & $r_{j+1}.r_j$ indicates concatenation of r_{j+1} & r_j.

If rank of sample X_i = k then it's rank will be represented as

$$R_i = \{ r_j = 0 \; ; \; 1 <= j < k \;\; \& \;\; k < j <= 2N+2 \\ = 1 \; ; \; j = k\}$$

Basically if rank of a sample = n then the nth bit from right in the rank register is set to 1 & all other bits are reset to 0. In other words, rank register is a flag register of size 2N+1 bits. Each bit flag is reserved for a possible value that a sample rank can take. It is set to "1" if the sample has that rank. All other bit flags for of the rank register of this sample will be reset to "0".

Associated with each processor P_i are 4 one bit flag registers U_i, V_i, W_i, Z_i.

Say, the window has samples X_{i-N} to X_{i+N+1} i.e. the two windows within this window are W_i & W_{i+1}. X_{i-N-2} & X_{i-N-1} are old samples being discarded & X_{i+N} & X_{i+N+1} are the new samples that have been taken in.

Operations in first computing unit

At the beginning of this operation, register a in processor P_j contains $X_{j+i-N-1}$, $1 <= j <= 2N+1$.

Compare with old & new samples

The four samples X_{i-N-2}, X_{i-N-1}, X_{i+N} & X_{i+N+1} are compared with contents of register a in processors P_1 through P_{2N}. The four comparators of the unit are used for following comparisons:

For j = 1 to 2N :	if $X_j > X_{i-N-2}$ then $U_j = 1$ else $U_j = 0$
For j = 1 to 2N :	if $X_j > X_{i-N-1}$ then $V_j = 1$ else $V_j = 0$
For j = 1 to 2N :	if $X_j > X_{i+N}$ then $W_j = 1$ else $W_j = 0$
For j = 1 to 2N :	if $X_j > X_{i+N+1}$ then $Z_j = 1$ else $Z_j = 0$

In the remaining two processors P_{2N+1} & P_{2N+2}, only one comparison takes place. The new sample Xi+N & Xi+N+1 are compared with each other & their relative ranks are determined. The Z flag is used for this. The other three flags need not be present in these two processors. The comparison is as follows :

```
If    $X_{i+N+1} > X_{i+N}$
then
      $Z_{2N+2} = 1$ & $Z_{2N+1} = 0$
else
      $Z_{2N+1} = 0$ & $Z_{2N+2} = 1$
```

The above algorithm basically defines job of each processor. At the end of this operation, each processor writes the contents of register a into register b. Register a of all the processors, shift to the right by two places & two new samples X_{i+N+2} & X_{i+N+3} corresponding to the next two windows W_{i+2} & W_{i+3}

Operations in the second computing unit

This computation unit is working on the previous window samples i.e. windows W_{i-1} & W_{i-2}. The rank register contains the ranks of windows previous to these (W_{i-3} & W_{i-4} combined) & they have to be updated. The flags contain the status of comparisons of the previous window (W_{i-1} & W_{i-2} combined).

Update Rank

All processors P_1 to P_{2N} execute the following algorithm in parallel

For j = 1 to 2N

```
    If $U_j = 1$ $R_j = R_j >> 1$
    If $V_j = 1$ $R_j = R_j >> 1$
    If $W_j = 1$ $R_j = R_j << 1$
    If $Z_j = 1$ $R_j = R_j << 1$
```

Can combine these to get a signal which would dictate the direction of shift.

Where $R_j >> 1$ represents a bit wise shift to the right by one & $R_j << 1$ represents a bit wise shift to the left by one.

This algorithm is good enough for rank updation of all elements except for the last two samples in processors P_{2N+1} & P_{2N+2}. Their ranks are computed as follows:

$$R_{2N+1} = R_{2N+2} = \text{NOT } (R_1+R_2+......+R_{2N-1}+R_{2N})$$

where NOT & '+' are logical NOT & OR operators, i.e. carryout a bitwise NOR on the rank registers of processors P_1 to P_{2N}. We call this rank prediction. We have predicted approximate rank register values of the two new samples. We arrive at the exact values, using the following procedure.

For both the samples X_{i+N} & X_{i+N+1} in processors P_{2N+1} & P_{2N+2} respectively.

```
If    $Z = 1$
then
      reset the rightmost 1 ~~bit=1,~~ of the rank register R, to 0
else
      reset the leftmost 1 ~~bit = 1,~~ of the rank register R, to 0
```

Now we have ranks of all the elements in the new window & are ready to find the medians.

Rank matching

We describe here the exception to the rule mentioned earlier. The median of the two windows are found as per following algorithm (the second computing unit is working on the previous two windows)

```
If    $(R_1 = \overset{A}{N+1})$ AND $(\overset{B}{R_{2N+2}} < N+1)$
                OR
      $(\overset{C}{R_{2N+2}} = N+1)$ AND $(R_1 \overset{D}{<} N+1)$
                OR
      $(R_1 < \overset{D}{N+1})$ AND $(\overset{B}{R_{2N+2}} < N+1)$
then
      $M_{i-2} = X_{R=N+2}$ ; $M_{i-1} = X_{R=N+2}$
Else
If    $(R_1 = N+2)$ AND $(R_{2N+2} > N+2)$
                OR
      $(R_{2N+2} = N+2)$ AND $(R_1 > N+2)$
                OR
      $(R_1 > N+2)$ AND $(R_{2N+2} > N+2)$
then
      $M_{i-2} = X_{R=N+1}$ ; $M_{i-1} = X_{R=N+1}$
Else
If    $(R_{2N+2} <= N+1)$
then
      $M_{i-2} = X_{R=N+2}$ ; $M_{i-1} = X_{R=N+1}$
Else
      $M_{i-2} = X_{R=N+1}$ ; $M_{i-1} = X_{R=N+2}$
```

$(R_1 <= N+1)$ AND
$A \cdot B + C \cdot D + D \cdot B$
$B(A + D) + CD$

The processors match the rank $N+1$ & N+2 & output the contents of register b as per above algorithm. The above algorithm can be easily implemented in a combinational logic.

This algorithm requires four comparators in each processor except for the last two processors. The algorithm can be used for large window sizes without affecting either the pipeline period or computation time or latency time. Another advantage of using the above mentioned rank representation, the rank matching logic & rank updation logic becomes extremely straight forward to implement, involving one or maximum two levels of combinational logic. The only limiting factor of this algorithm is the size of the rank register R which takes up 2N+2 bits.
The processor pipeline working frequency need not be the same as the sampling rate. The system as such can work at half the sampling rate frequency. Figure 7 shows a sequence of operations for Design II.

However, one area that needs to be further improved is the rank updation logic of the last two elements. These two ranks can only be updated after all the other $2N$ ranks are updated, thus forcing this part of the algorithm to run serially.

IV. 2 Dimensional filters

Given a 2 dimensional sequence with size of $M \times M$, a 2 dimensional window W_{ij} with size $N \times N$ is used for the operation of the rank order filtering. When this 2 dimensional window moves horizontally across a stripe of 2 dimensional data sequence with size $M \times M$, the 2 dimensional rank filter can be interpreted as a 1 dimensional rank order filtering with a 1 dimensional window (of length N^2) moving across an expanded 1 dimensional sequence (of length NM) & each movement of the window jumps multiple N data samples (see figure 8).

By using the same technique as in Design II, the architecture can be easily extended for 2 dimensional filters. A brief description with reference to the above discussion follows.

Since N samples are being processed at a time, architecture for 2 dimensional filters will require N^2 processors, with 2N comparators in the first N^2-N processors. The algorithm will require to perform 2N comparisons in the N^2-N processors : N comparisons with N old samples & remaining N with N new samples. To keep track of these 2N comparisons, each processor will need 2N 1 bit flags. The last N processors, which will basically receive the N new samples will need N-1 comparators & equal number of 1 bit flags each.

The comparison of old & new samples can be carried out as per the algorithm of Design II. This will give the ranks of the overlapping N^2-N samples & relative ordering of the N new samples. The rank updating & predicting algorithm can be extended for N samples and can be used here. However, the elaborate rank matching algorithm of Design II is not needed in this case. All the processors will match the desired order statistic rank with their rank registers & output their sample values if a match occurs (similar to the rank matching algorithm of Design I).

Note that this algorithm takes as input N samples at a time & processes them all in parallel, resulting in very high speed 2 dimensional median filter. Also note that the latency time & pipeline period are the same as that of the one dimensional filter. No PISO logic is required as in [4], no cascading is required for maximum throughput & the availability of data in parallel is fully exploited. However, the size of the rank register increases at a rate N^2 with increase in N.

V. Conclusion

In this paper, we have described two new semisystolic array architectures & algorithms for 1 and 2 dimensional median filters. A high sample rate is achieved by processing multiple inputs simultaneously & efficiently pipelining the computations. The algorithms can be easily extended over to any order statistic filters.

References

[1] J. P. Fitch, E. C. Coyle & N. C. Gallagher, "Median filtering by threshold decomposition", IEEE Trans on Circuits, Systems, vol. CAS 34, 1987, pp. 553-559.

[2] N. C. Gallagher, Jr. & G. L. Wise, "A theoretical analysis of the properties of median filters", IEEE Trans on Acoust., Speech, Signal Processing, vol. ASSP 29, Dec 1981, pp. 1136-1141.

[3] D. S. Richards, "VLSI median filters", IEEE Trans on Acoust., Speech, Signal Processing, vol. ASSP 38, Fan 90, pp. 145-153.

[4] J. N. Hwang, J. M. Jong, "Systolic architecture for 2-D rank order filtering", Int'nl Conf. on App. Specific Array Processors, Sept 1990, pp. 90-99

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

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

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

[8] C. Chakrabarti, "High sample rate systolic architectures for median filters".

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

Figures

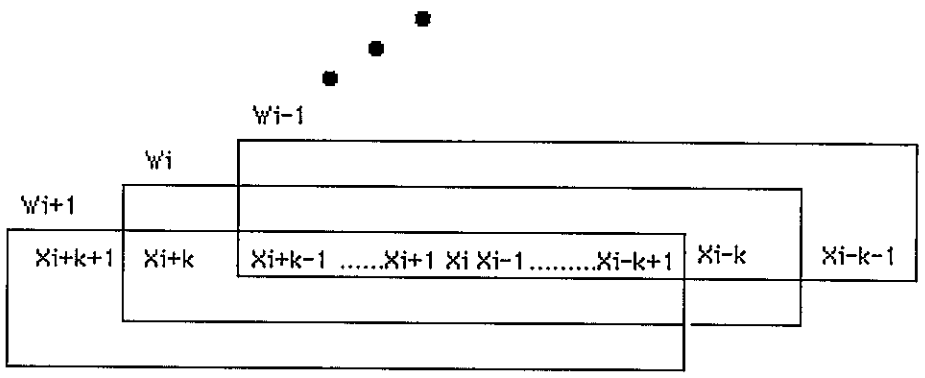

Fig 1. Window W slides over the input samples. Window Wi is obtained from Wi-1 by deleting Xi-k-1 & inserting X i+k

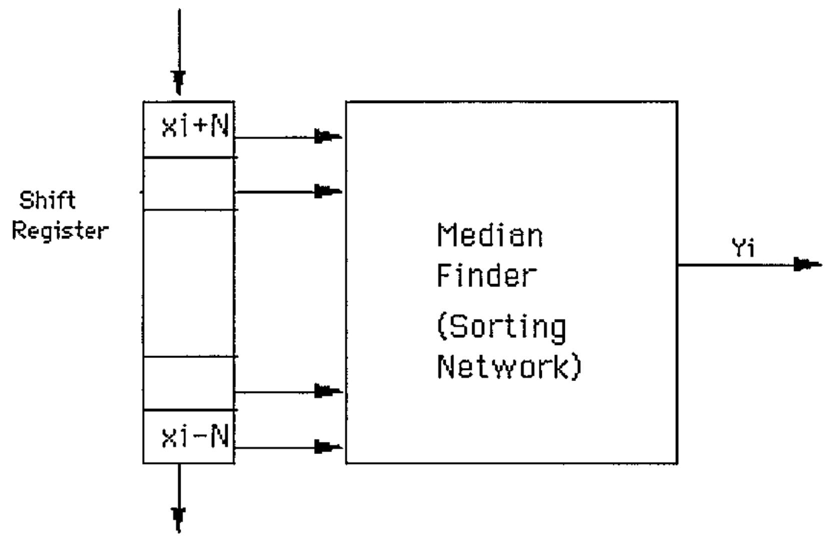

Fig 2 Design using a median finder with unsorted values

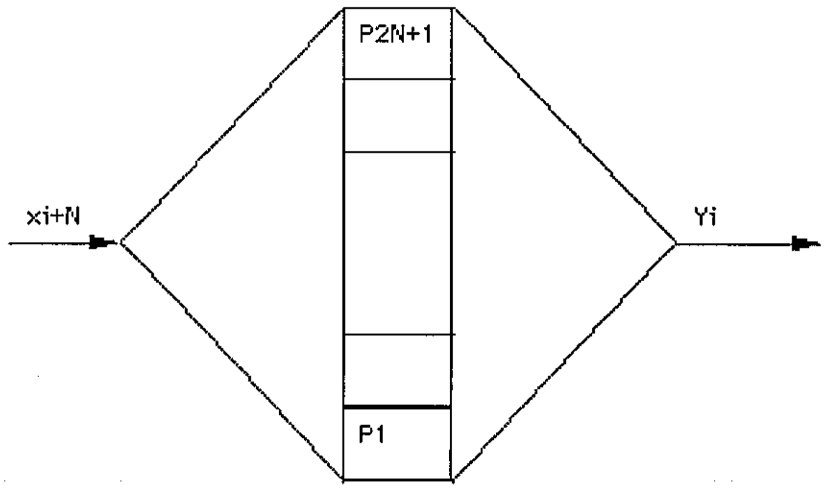

Fig 3 . Design that maintains the value in sorted order.

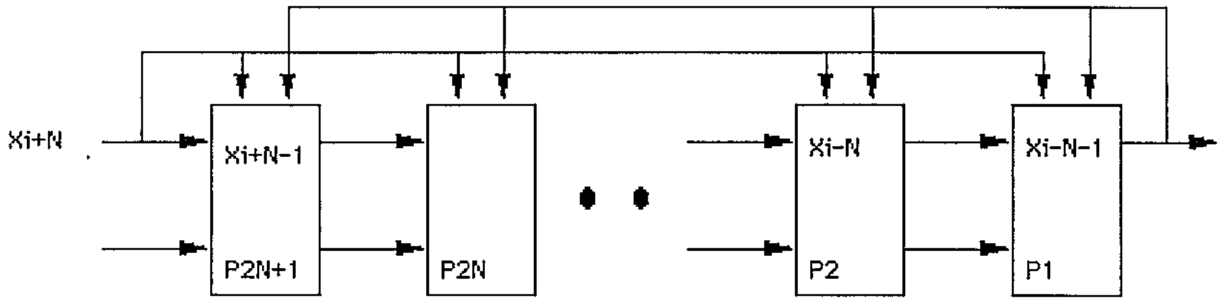

Fig 4 (a)

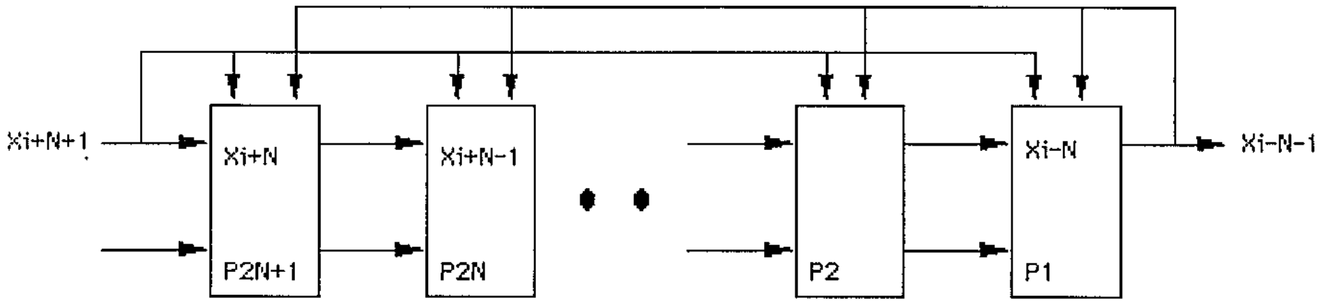

Fig 4 (b)

Fig 4. (a) Processor array with window Wi-1. (b) Processor array with window Wi.

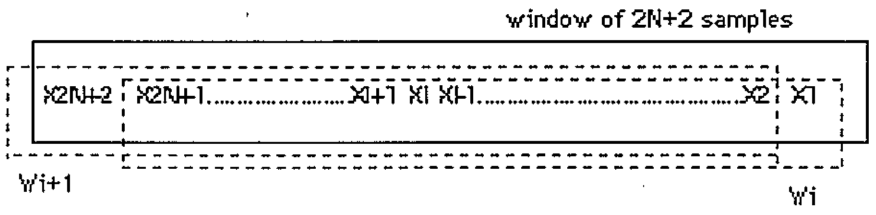
window of 2N+2 samples
X2N+2
X2N+1......................XI+1 XI XI-1..X2
X1
Wi+1
Wi

Fig 5. A window of 2N+2 samples consisting of two overlapping windows of 2N+1 samples each.

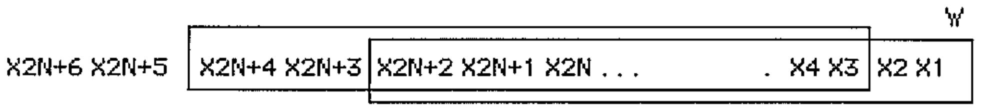

Fig 6.Window W slides over the input samples covering two windows at a time.

Proc	Reg a	Compare	Shift a	Reg b	U	V	W	Z	Reg R	Updated Reg R	Final R P5, P6	Median
	8 (old sample)		5									
	6 (old sample)		3									
P_1	5	5: 8,6,4,2	7	8	1	1	1	1	100000	100000		
P_2	3	3: 8,6,4,2	1	6	1	1	0	1	010000	001000		M_{i-2}
P_3	7	7: 8,6,4,2	4	5	1	1	0	1	001000	000100		M_{i-1}
P_4	1	1: 8,6,4,2	2	3	0	1	0	1	000100	000010		
P_5	4	4: 2	6	7				1		010001	010000	
P_6	2	2: 4	4	1				0		010001	000001	
	6											
	4											

Fig 7 (a).

	5		7									
	3		1									
P_1	7	7: 5,3,6,4	4	5	0	0	1	1	000100	010000		
P_2	1	1: 5,3,6,4	2	3	0	0	0	1	000010	000100		M_{i+1}
P_3	4	4: 5,3,6,4	6	7	0	1	1	1	010000	100000		
P_4	2	2: 5,3,6,4	4	1	0	0	0	0	000001	000001		
P_5	6	6: 4	3	4				1		001010	001000	M_i
P_6	4	4: 6	5	2				0		001010	000010	
	3											
	5											

Fig 7 (b).

Fig 7. Sequence of operations that take place simultaneously in the two computation units. The input sequence being processed is 8, 6, 5, 3, 7, 1, 4, 2, 6, 4, 3, 5.......... Window sizes of 5 (i.e. a combined window size of 6) are being processesd $W_i = \{5, 3, 7, 1, 4\}$. (a) Fig (a) is the snapshot of the array at time t = T. At this time instance, the first computing unit will be processing windows W_i & W_{i+1}. The second computing unit will output median values of windows W_{i-1} & W_{i-2}. (b) Fig (b) is the snapshot of the array at time t = T+1.

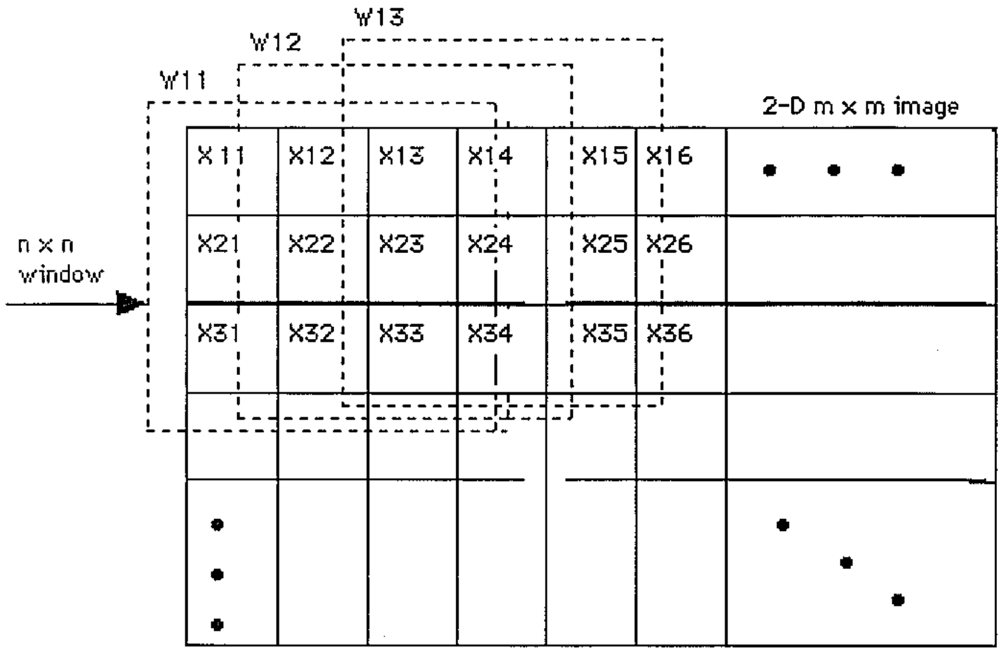

Fig 8 (a)

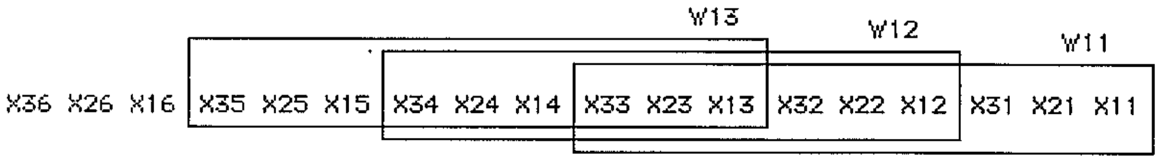

Fig 8 (b)

Fig 8. (a) A 2-d windowed image sequence. (b) Conversion from 2-D windowed sequence to 1-D windowed sequence with multiple data movements at each window. All the new data samples are available at the same time.