5. Nonlinear Digital Signal Processing

5.1 Introduction

In the early development of signal processing, linear methods were the primary tools. Their mathematical simplicity, the well developed theoretical background and the existence of some desirable properties make them easily design and implement. Moreover, linear methods offered satisfactory performance in many applications. However linear solutions have poor performance in the presence of noise that is not additive as well as in problems where system nonlinearities or non-Gaussian statistics are encountered. For such reasons, nonlinear signal processing methods became more and more important in the last two decades. The purpose of this chapter is to summarise some knowledge about nonlinear signal processing. The emphases will be on such algorithms that can be easily implemented in parallel form. The chapter does not give a detailed description of all important methods. Instead, it deals with only some special algorithms and systems: there will be two main parts of the chapter. The first will deal with nonlinear filters, mainly with order statistic filters as well as with nonlinear filters with the linear in-the-parameter structure (such as Volterra filters, RBF filters, etc.); the topic of the second part will be (nonlinear) signal processing using neural networks.

5.2 Nonlinear filters

A filter can be defined in a mathematical sense as an operator $L(.)$, which maps a signal x into an other signal y as follows:

$$y=L(x) \qquad (5.2.1)$$

When $L(.)$ is a linear operator the filter is linear, for nonlinear filters the operator is not linear. In conventional linear filter's operation the output value of the filter is simply some linear combination of the input signal, i.e. no nonlinear operation

on the input signal is performed. Such filters are therefore very simple to implement and are very popular in many applications. It is well known, however, that the applications of linear filters to real word problems like time series prediction, channel equalization, etc. are sub-optimum, and substantial improvements can be achieved by using nonlinear methods. For nonlinear filters the superposition or proportionality principles or both does not apply, the output of a nonlinear filter cannot be obtained by convolving the input with the filter's impulse response function. For this reason different families of nonlinear filters have been studied, each of which has a different characterization which makes the filters in this class amenable to some form of mathematical analysis.

5.2.1 Order statistic filters

One of the most popular families of nonlinear filters for noise removal is order statistic filters. Order statistics play important role in the statistical data analysis and especially in the robust analysis of data contaminated with outlying observations. Their robustness properties made them suitable for time series analysis, and for digital signal and image processing applications.

Order statistic filters form a whole family. There exist several filters which are members of this family, e.g. median filters, the stack filters, the median hybrid filters, etc.

Order statistic can be defined as follows: let the random variables x_1, x_2,…, x_n be arranged in ascending order of magnitude and then written as

$$y(n) = h_0 + \sum_{k=1}^{\infty} \bar{h}_k[x(n)] \tag{5.2.2}$$

x_j is the so called j-th order statistics.

A very important order statistic is the median. Median filter, which was first suggested by Tukey in 1971 [Tuk71], represents one of the simplest order statistic filter. The median of n observations x_i i=1,2,…,n is denoted by $med(x_i)$ and it is given by:

$$med(x_i) = \begin{cases} x_{\nu+1} & n = 2\nu + 1 \\ x_{\nu} + x_{\nu+1} & n = 2\nu \end{cases} \tag{5.2.3}$$

A one-dimensional median filter of size n=2ν+1 is defined by the following input output relation:

$$y_i = med\,(x_{i-\nu}, \ldots x_i, \ldots x_{i+\nu}) \qquad i \in \mathbf{Z} \tag{5.2.4}$$

Its input is the sequence x_i and its output is the sequence y_i, so it is called moving median or running median.

A two-dimensional median filter has the following definition:

$$y_{ij} = med\left\{x_{i+r,j+s};\ \ (r,s) \in A\right\} \qquad (i,j) \in \mathbf{Z}^2 \tag{5.2.5}$$

The set $A \subseteq \mathbf{Z}^2$ defines the neighbourhood of the central input element (i,j) (in image processing the central pixel) and it is called the filter window. If the input image is of finite extent NxM, $1 \le i \le N$, $1 \le j \le M$, definition (5.2.5) is valid only for the interior of the output image, i.e. for those (i,j) for which:

$$1 \le i + r \le N, \qquad 1 \le j + s \le M \qquad (r,s) \in A \tag{5.2.6}$$

At the border of the image (5.2.6) is not valid. There are two approaches to solve this problem. In the first one the filter window A is truncated in such a way so that (5.2.6) becomes valid and definition (5.2.5) is used again. In the second approach the input sequence is appended with sufficient samples and the definition (5.2.50 is applied for $1 \le i \le N$, $1 \le j \le M$.

So median smoothing is a filtering operation, widely used in digital signal and image processing, which involves replacing each sample value with the median of the values found within some neighbourhood of itself.

Median filters can be applied successfully for elimination of outliers, spurious values caused by noise or other technical error. Median filtering has several properties that are particularly useful in image processing. One is that the operation does not introduce intermediate pixel values not found in the original image, as convolution methods may and hence preserves sharp region boundaries. Also, since the median of a group of numbers is generally insensitive to the presence of a small number of outliers, median smoothing is not subject to the problem of artifact ringing, the propagation of an error value through a region of an image. For this reason median filtering is sometimes used as a preprocessing step before an image is filtered by other means.

Median filtering can be generalized by considering order statistics other than the median; while the median of n numbers is the one having rank $(n+1)/2$ (for odd n), we can consider asking for the element having some arbitrary rank r.

5.2.2 Homomorphic filters

In many applications signals are combined in a rather complicated way. Convolved signals are encountered in seismic signal processing, digital speech processing, digital echo removal and digital image restoration. Signals combined in a nonlinear way are encountered in digital signal processing for communication systems and in digital image filtering. Classical linear processing techniques are not so useful in those cases because the superposition property does not hold any more. Therefore, a special class of filters has been developed for the processing of convolved and nonlinearly related signals. They are called homomorphic filters. Their basic characteristic is that they use nonlinearities (mainly the logarithm) to transform convolved or nonlinearly related signals to additive signals and then to process them by linear filters. The output of the linear filter is transformed afterwards by the inverse nonlinearity.

Homomorphic filtering has found many applications in digital image processing. It is recognised as one of the oldest nonlinear filtering techniques applied in this area. The main reason for its application is the need to filter multiplicative and signal-dependent noise. Linear filters fail to remove such types of noise effectively. Furthermore, the nonlinearity (logarithm) in the human vision system suggests the use of classical homomorphic filters. Homomorphic filtering can also be used in image enhancement. Object reflectance and source illumination contribute to the image formation in a multiplicative way. Ideally, the source illumination is constant over the entire image. However, in many practical cases, e.g., in outdoor scenes, source illumination is not constant over the entire scene. Therefore, it can be modelled as noise in the low spatial frequencies. If this noise is removed, the object reflectance is enhanced. Homomorphic filtering has found various practical applications, e.g. in satellite image processing and in the identification of fuzzy fingerprints. Homomorphic filtering has also found several applications in other areas, e.g. in speech processing and in geophysical signal processing. In the following a short introduction to the theory of homomorphic filters will be given.

Algebraic theory of homomorphic signal processing

One very important property of linear filters is the superposition property, which allows them to process signals that have been added. However, sometimes signals are combined in other ways, e.g. in a multiplicative way:

$$y(i)=x(\mathrm{i})\, s(i) \tag{5.2.7}$$

In this case the superposition property does not hold any more and the linear filters are of limited use. A useful idea is to map them to additive signals and then to process them in a linear way.

Let * and ⊗ be two operations in the input signal space and in the output signal space, respectively. Suppose that a homomorphic system H transforms the input signal space to the output signal space, as is shown in Figure 5.2.1.

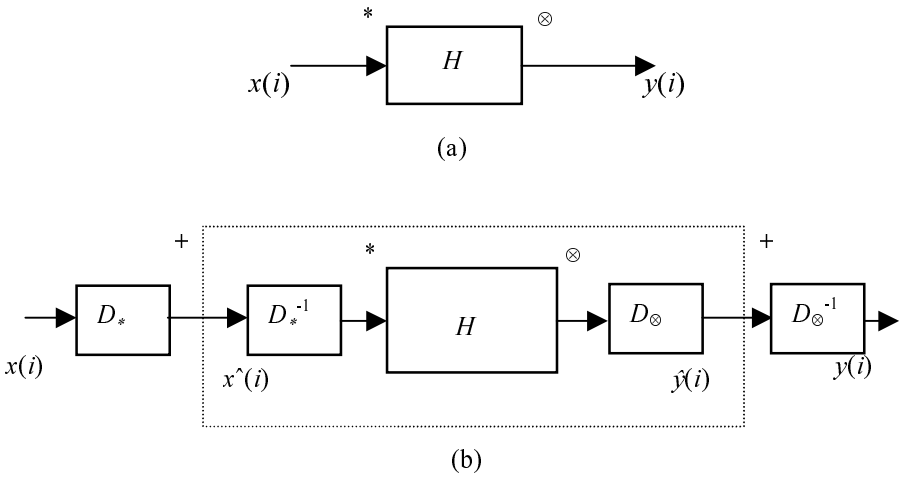

Fig 5.2.1: Definition of a homomorphic system; (b) Canonical decomposition of a homomorphic filter.

Two isomorphisms D_*, $D_\otimes$ can be found, which transform the input and the output signal spaces to the standard additive signal space. Let also D_*^{-1}, $D_\otimes^{-1}$ be their inverse transformations. In this case the homomorphic system H can take the form of figure 5.2.1.*b* because the transformations are cancelled by their inverses. The cascade of D_*^{-1}, H, $D_\otimes$ is a linear system that operates on the additive signal space. So the cascade of these three operators can be replaced by a linear operator.

If this linear system is denoted by L, the homomorphic system takes its canonical decomposition form. Let two signal vectors $x_1(i)$, $x_2(i)$ be combined by the operation *:

$$x(i)=x_1(i)*x_2(i) \tag{5.2.8}$$

If $x(i)$ passes through the homomorphic system H, it is transformed as follows:

$$\hat{x}(i) = D_*[x_1(i)] + D_*[x_2(i)] = \hat{x}_1(i) + \hat{x}_2(i) \tag{5.2.9}$$

The system L is linear. Therefore:

$$\hat{y}(i) = L[\hat{x}(i)] = L[\hat{x}_1(i) + \hat{x}_2(i)] = \hat{y}_1(i) + \hat{y}_2(i) \tag{5.2.10}$$

The output $y(i)$ of the homomorphic system H is given by:

$$y(i) = D_{\otimes}^{-1}[\hat{y}_1(i) + \hat{y}_2(i)] = y_1(i) \otimes y_2(i) \tag{5.2.11}$$

Thus the homomorphic system satisfies the following generalised principle of superposition:

$$H[x_1(i)*x_2(i)]=H[x_1(i)] \otimes H[x_2(i)] \tag{5.2.12}$$

It is clear that after the choice of the characteristic functions D_*, $D_{\otimes}$, the filtering properties of the homomorphic filter lie in the appropriate choice of the linear filter L.

5.2.3 Polynomial filters

In many problems of digital signal processing it is necessary to introduce nonlinear systems. For example; it is well known that in detection and estimation problems, nonlinear filters arise in the case where the Gaussian assumption is not valid or the noise is not signal independent and and/or additive.

A possible way to describe the input-output relation in a nonlinear filter that is amenable to characterization, analysis, and synthesis is to use a discrete Volterra series representation. The Volterra series can be viewed as a Taylor series with memory. Although the class of nonlinear systems that can be represented by the Volterra series expansion is limited, the Volterra series is one of a few representations that are easily amenable to characterization and analysis. In many cases, a nonlinear system can be represented by a truncated version of the Volterra series, which results in a simpler representation and requires a limited amount of knowledge of higher order statistics.

Filters based on the Volterra series and another representation called Wiener series will be referred to as polynomial filters. Such filters can be useful in many situations in which linear processors are known to be suboptimal; such as the equation of nonlinear channels. Polynomial filters can also be applied for compensating typical nonlinearities that affect adaptive echo cancellers; because in practical systems various kinds of nonlinearities are present in the echo path or in the echo replica, such as transmitted pulse asymmetry, saturation in transformers, nonlinearities of data converters etc.

In image processing a special class of polynomial filters have been used for image enhancement, edge extraction; nonlinear prediction, and nonlinear interpolation of TV image sequences. Polynomial filters have also been used in modeling geophysical and sonar channels with nonlinearities.

Daily stock returns are another example of real data that exhibits nonlinearities in its behavior.

If the input-output relation is restricted only to the third term of a Volterra processor, the system becomes a quadratic filter. It is very common to reduce a nonlinear system to the quadratic part. In addition, quadratic filters appear in many problems of digital signal processing, such as in the optimal detection of a Gaussian signal in Gaussian noise as well as texture discrimination. Quadratic filters are the simplest nonlinear, time-invariant systems, and their mathematical properties, harmonic representation, and efficient implementation have been considered.

Definition of polynomial filters

In this chapter, a class of nonlinear, one-dimensional, shift-invariant systems with memory based on the discrete-time Volterra series is considered. The input-output relation satisfied by these systems, when the discrete independent variables are defined on a finite support, is then given by (5.2.13):

$$y(n) = h_0 + \sum_{k=1}^{\infty} \overline{h}_k[x(n)] \qquad (5.2.13)$$

where $x(n)$ and $y(n)$ denote the input and output, respectively, h_o is a constant term, and $\overline{h}_k[x(n)]$ is defined by (5.2.14).

$$\overline{h}_k[x(n)] = \sum_{i_1=0}^{N-1} \ldots \sum_{i_k=0}^{N-1} h_k(i_1,\ldots,i_k)x(n-i_1)\ldots x(n-i_k) \qquad (5.2.14)$$

Note that in (5.2.14) for k=1 the term $h_1(i_1)$ is the usual linear impulse response, and the term $h_k(i_1, \ldots i)$ can be considered as the finite extent k-th order impulse response, which characterizes the nonlinear behavior of the filter. In a Volterra filter of order K the upper limin in (5.2.13) is replaced by K.

The quadratic filter is given by the third tem of (5.2.13) or the second term of (5.2.14) and is described as:

$$y(n) = \sum_{i_1=0}^{N-1} \sum_{i_2=0}^{N-1} h_2(i_1, i_2) x(n-i_1) x(n-i_2) \qquad (5.2.15)$$

or in a compact format:

$$y(n) = \sum_{i=1}^{N} \sum_{j=1}^{N} h_{i,j} x_{n-i+1} x_{n-j+1} \qquad (5.2.16)$$

The input output relationship of a quadratic filter can be written in a vectorial form:

$$y_n = \mathbf{x}^T \mathbf{H} \mathbf{x} \qquad (5.2.17)$$

where **H** is a symmetric matrix, i.e. $\mathbf{H}=\mathbf{H}^T$ We call this matrix the kernel of the filter.

There are some special versions of the quadratic filters. Two such special quadratic filter structures are shown in figure 5.2.2.

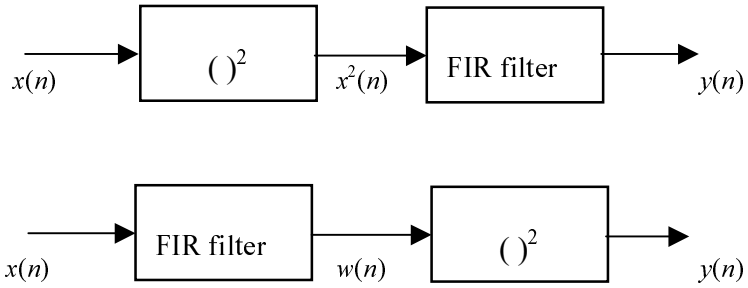

Fig. 5.2.2 Special forms of quadratic filters

In these structures FIR (linear) filters are used in combination of a nonlinear static network, which implements a square function.

Filters similar to the Volterra filters can be implemented using neural networks too. As neural networks can implement almost any continuous nonlinear static mappings, if we replace the square function block in the structures of figure 5.2.2 by a neural network, special NFIR nonlinear filter structures can be obtained. Some further possibilities about using neural nets for implementing nonlinear filters will be discussed in section 5.3.

5.2.4 Implementation of nonlinear filters

One of the main problems in linear as well as nonlinear filtering/signal processing is the computational complexity and the execution speed of signal

processing routines, especially when real-time processing is required. This problem mainly results from the amount of data to be processed (e.g. in image processing a single 512x512 8-bit image is 256 K bytes), and the number of operations required per output data (output pixel in image processing). Such operations are comparisons, additions, multiplications with constants and nonlinear function evaluation. There are two solutions for increasing the speed of processing. The first one is to construct fast algorithms, which can be run on (rather high-speed) general purpose computers. The second one is to use special signal processors. In many cases – mainly for real-time processing – only the application of special parallel architectures can fulfil the speed requirements. In this chapter first some algorithms - needed by nonlinear filters - will be discussed shortly, then some (parallel) architectures will be presented which are suitable for nonlinear filtering.

Algorithms

The main tasks in order statistic filtering are sorting and selection. Sorting and selection are classical topics of computer science. Their performance, usually measured in number of comparisons, has been thoroughly analysed. Several such algorithms (e.g. MERGESORT, QUICKSORT) have become classical and their description and analysis can be found in the basic computer science textbooks such as [Knu73]. The classical sorting and selecting algorithms can be very easily adapted to filtering applications. However, sorting and selection (e.g. median calculation) are running operations. Thus the filter windows centered at neighbouring data overlap. This window overlapping cannot be taken into advantage if classical sorting and selection algorithms are used. The running and selection algorithms most suitable for median filters are based exactly on the window overlapping. One of the running algorithms was proposed by Huang et al. [Hua79] for two- dimensional (image) median filtering.

Let us suppose that the filter window has mxn pixels and that it is moving row-wise. When the window is moved from position (i,j) to the position $(i,j+1)$, mn-2n pixels remain unchanged, n must be discarded and n new pixels must be inserted. Huang's algorithm is based on the local grey level histogram of the nm pixels and on its adaptation, as the window moves.

The steps of the algorithm are as follows:

1. Calculate the local histogram of the nm pixels in the first window and find the median. Count the number v of pixels with grey levels less than that of the median.

2. Move to the next window by deleting n pixels and inserting the n new pixels. Update the histogram. Count the number v of pixels whose grey level is less than the median of step 1.
3. Starting from the median of the previous window, move up/down the histogram bins one at a time if the count v is smaller/greater than $(mn+1)/2$ and update v until the median is reached.
4. If the line end has been reached, move to the next line and go to step 1. Otherwise go to step 2.

This algorithm requires only $2n$ computations for an output point, whereas the QUICKSORT algorithm requires $O\ (2n^2\ log\ n)$ comparisons if mxn data points (pixels) are processed and if $m \cong n$. Thus the running median algorithm is much faster than the QUICKSORT used in median filtering.

Another approach for running median calculation is based on partitioning of the pixels in the filter window in two subsets [Ahm87]. Let the window size be mxn and $W(i,j)$ be an mxn matrix containing the modified version of the data x_{kl} in filter window centered at the pixel (i,j), such that the elements of each column of $W(i,j)$ are in ascending order. The last m-1 columns of the window $W(i,j)$ are also included in the window $W(i,j-1)$. Only the rightmost column of $W(i,j)$ differs. This column can be obtained from the corresponding column of $W(i-1,j)$ by deleting the pixel $x(i-(m+1)/2,\ j+(n-1)/2)$ and by inserting the pixel $x(i+(m-1)/2,\ j+(n-1)/2)$ in such a way so that the rightmost column of $W(i,j)$ is ordered. The median at the pixel $(i,j-1)$ can be found in the following way. The pixels in the window $W(i,j-1)$ are partitioned into two subsets. Subset S_1 contains $(mn+1)/2$ elements and subset S_2 contains the rest $(mn-1)/2$ elements that are greater than or equal to the largest element of S_1. The median at $(i,j-1)$ is the largest element in S_1. The median at position (i,j) can be obtained by updating $W(i,j-1)$ to the new window $W(i,j)$. This can be done in the following steps:

1. Discard the leftmost column of $W(i,j-1)$.
2. Obtain the rightmost column of $W(i,j)$ from the corresponding column of $W(i-1,j)$ as described previously.
3. Include the elements of the new column that are less than or equal to the median at position $(i,j-1)$ into the subset S_1 and the rest of them to the subset S_2.
4. (*) If the number of elements in S_1 is just one more than the number of elements in S_2, final partitioning has been obtained. (**) If the number of elements in S_1 is less than $(mn+1)/2$, the smallest elements of S_2 are pushed

to S_1 until the number of elements of S_1 becomes equal to $(mn+1)/2$. (***) If the number of elements in S_1 is greater than $(mn+1)/2$, the largest elements of S_1 pushed into S_2 until the number of elements of S_1 becomes equal to $(mn+1)/2$. In all cases the new median of the largest element of S_1.

The algorithm involves basically three operations. The first one, which deletes an element from a list of m numbers and inserts a new number, requires m comparisons in the worst case. The second operation is to search for the position of the element in the rightmost column, which is greater than or equal to the old median. This operation requires at most m comparisons. The third operation is to update the two subsets. It involves the calculation of the smallest or the largest number along the border, i.e. from a set of n numbers. This process requires in the worst case $(2n-1)(m+1)/2$ comparisons. Thus in the worst case the number of operations required by this algorithms of the order $O(n2)$. However, it has been found that the actual number of comparisons in practical cases is much less and that the algorithm is 15-25% faster than the Huang's algorithm for window sizes up to 15x15.

Another approach to fast median filtering is to use the binary representation of the image pixels in the filter window as it is seen in figure 5.2.3. Rows represent pixels and columns represent bit planes. Let each pixel k bits. The least significant bit plane is B_k and the most significant one is B_1. Lexicographic sorting uses binary number representation. In the first step, the sequences are sorted according to the most significant bit B_1. In the second step, they are ordered according to the next significant bit and the result of the first step. This process continues until the least significant bit is reached after k steps. The median is at row $v+1$ when the ordering is finished. Lexicographic ordering requires kn binary comparisons.

i	B_1	B_2	B_3	B_4	
1	0	0	1	1	x_1=3
2	1	1	0	0	x_2=12
3	1	0	0	0	x_3=8
4	1	0	1	1	x_4=11
5	0	1	1	1	x_5=7

Fig. 5.2.3 Binary representation of pixels having k=4 bits each in a filter window of length 5

A modification version of this algorithm is proposed in [Dan81] for selecting the i-th order statistic. Let n_0 be the number of pixels having the most significant bit B_1 equal to zero. If $n_0=i$ than the i-th order statistic belongs to the partition of pixels that have $B_1=0$. The rest of the pixels can be discarded. If $n_0<i$, than the i-th order statistic belongs to the partition of pixels that have $B_1=1$. Let us consider the first case, i.e. $n_0=i$. Let n_{01} be the number of pixels having 01 as the two most significant bits. If $n_0-n_{01}<i$, then the i-th order statistic lies in the partition of pixels having 01 as the two most significant bits. This procedure can be repeated until all bit planes are examined. Some pixels are discarded at each step, until only one pixel remains at the end. This pixel is the i-th order statistic. Thus this algorithm determines the bit representation of the i-th order statistic successively. The whole procedure can be described by a binary tree similar to that shown in figure 5.2.4. Other algorithms using bit representation of the image pixels for the calculation of the median or other order statistics can be found for example in [Ata80], [Hoc88] and [Hoc89].

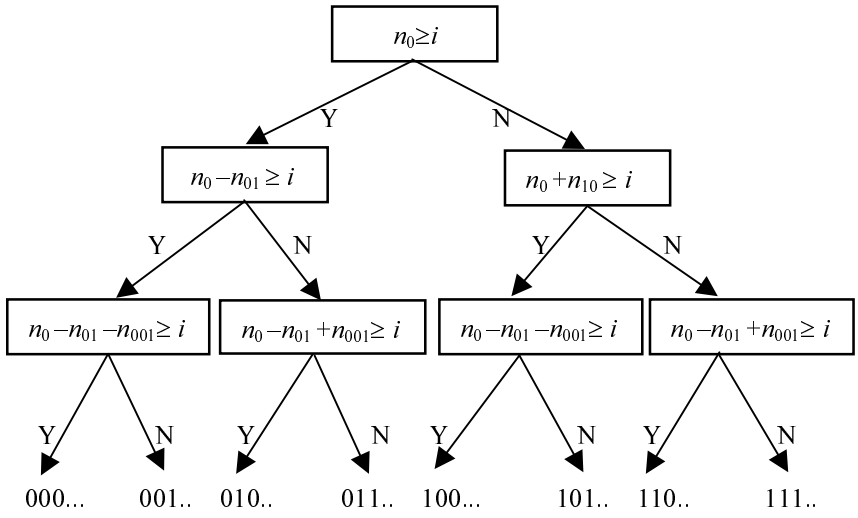

Fig. 5.2.4 Binary tree for the selection of the i-th order statistic

Fast structures for median and order statistic filters

So far we were mostly interested in the construction of fast algorithms for median and order statistics filtering mainly by minimizing the number of comparisons

required. No special attention has been paid for data movements or for the complexity of the final decision structure. However, sorting algorithms can be constructed that have a very regular structure and which can be used for the development of very regular and parallel architectures. Such algorithms are the sorting networks. Such a network is shown is figure 5.2.5.

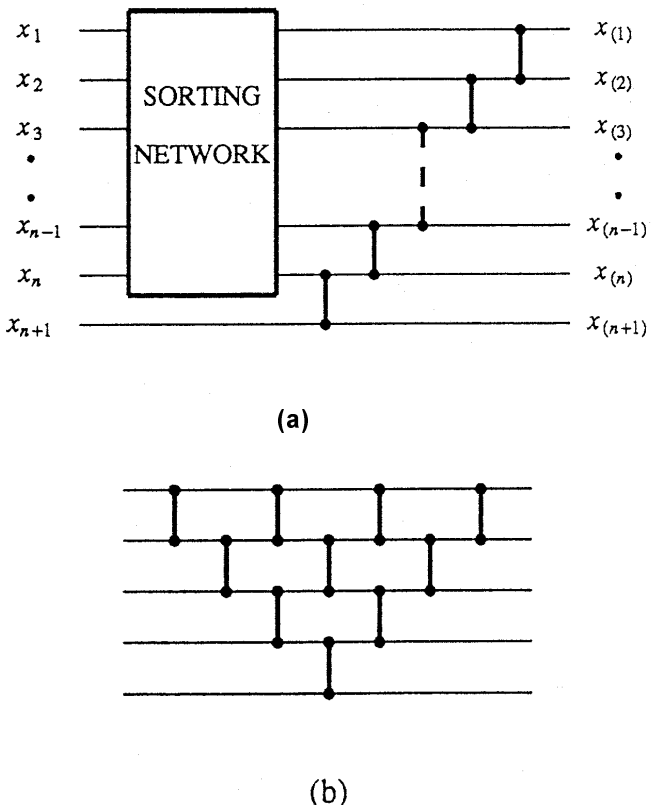

Fig. 5.2.5 (a) Sorting by insertion; (b) Sorting network for n=5

The data enter at the left side of the network. Comparators are represented by vertical connections between the network lines. Each comparator can shuffle its inputs if necessary, so that the large number appears on the lower output of the comparator and the small number appears to its upper output. Sorting networks have many advantages apart from their regularity and parallelism. They always

do in-place calculation, i.e. the data can be maintained in the same n locations. Also, no storage of the results of previous comparisons is required. One simple way to construct a sorting network is by *insertion* as it is shown in figure 5.2.5 (a). The (n+1)-st element can be inserted to its proper place, after n elements have been sorted. The sorting network for n=5 numbers is shown in figure 5.2.5 (b).

Sorting networks can be constructed using *selection.* In this case first the maximum element $x_{(n+1)}$ is calculated. Repeated application of this maximum selection lead to the bubble sorting network. If parallelism is allowed, straight insertion also leads to bubble sorting network. Sorting networks can be constructed that require minimum number of comparators, if hardware restrictions exist. Such a network for sorting n=9 numbers requires 25 comparators. A network, that can be easily implemented in hardware is the odd-even transportation network (figure 5.2.6). It requires $n(n-1)/2$ comparators in a brick-like pattern. Its simplicity lies in the fact that only two operations alternate.

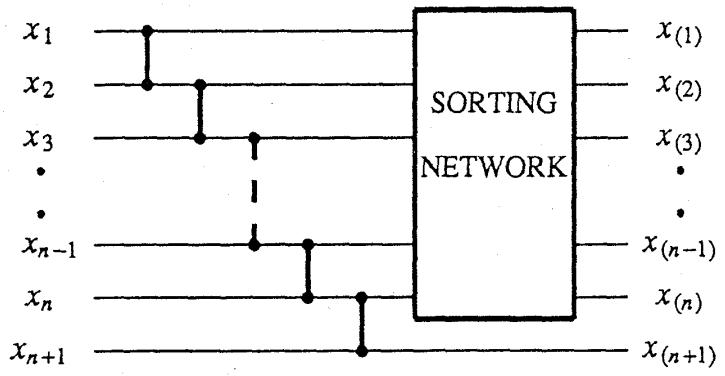

Fig. 5.2.6 Sorting by selection

This network has been proposed for VLSI implementation of a one-dimensional median filter of length n=5, which is shown in figure 5.2.7. The two blocks used in this implementation are delay elements (T) and compare-and-swap circuits (CS). A smaller odd-even transportation block for n=3 is shown in figure 5.2.8 as a building element of a sorting network for n=9 numbers. The odd-even transportation network has been proposed for the implementation of a 3x3 nonlinear filter module, as it is shown is figure 5.2.9. This filter module can be

used for the implementation of a variety of nonlinear digital filters (e.g. median filters, L-filters, morphological filters, homomorphic filters).

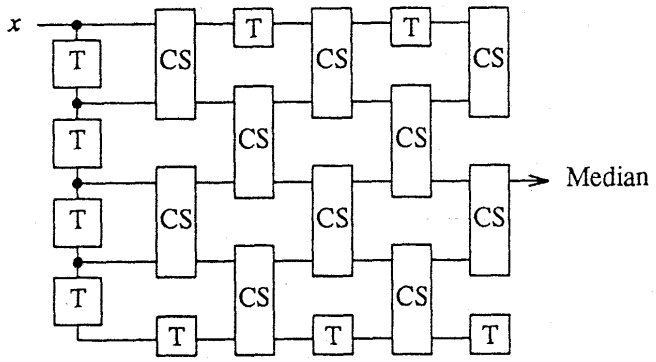

Fig. 5.2.7 A VLSI structure for median filtering for n=5 filter size

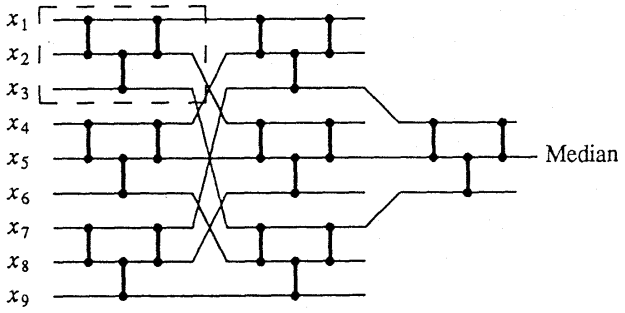

Fig. 5.2.8 Median filtering structure for n=9 numbers, using the odd-even transportation network for n=3

Morphological processors

The main characteristic of the morphological operations is their simplicity. The basic operations involved are the erosion and dilation. If binary images are

involved, the following definition of erosion and dilation can be used for their implementation on a computer:

$$X - B^s = \{z \in E : B_z \cap X \neq 0\}$$

$$X + B^s = \{z \in E : B_z \cap X \neq 0\}$$

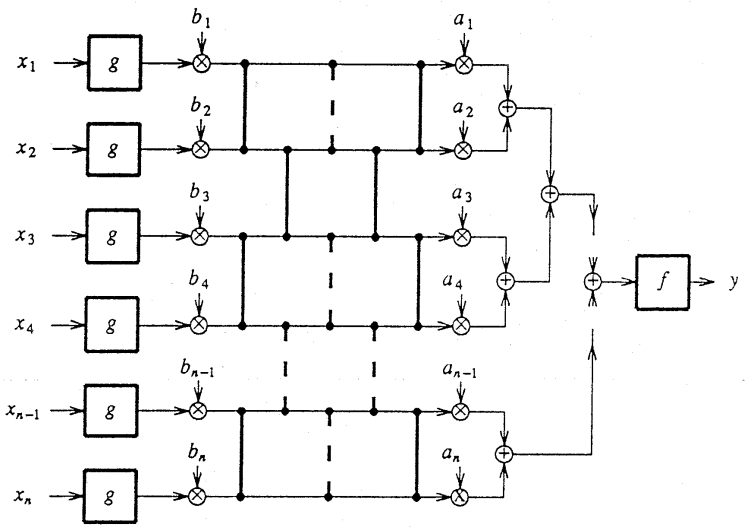

Fig. 5.2.9 General nonlinear filtering module

where z is arbitrary position on the image plane and B_z is the translate of the structuring element B in that position. Both definitions of erosion and dilation suggest that local operations around the position $z=(i,j)$ of the image plane are required for the calculation for both erosion and dilation. Thus cellular computer architectures can be used for the implementation of morphological operations.

Such cellular array architecture is operating on a 3x3 image neighbourhood is shown in figure 5.2.10. Each of its cell processors has the structure of figure 5.2.11, each cell has a register for storing the state of the cell and a logic module, which calculates the new state of the module as a function of its current state and the states of the neighbouring cells. An example of cellular image processor is

CLIP (Cellular Logic Image Processor). CLIP4 has an array of 96x96 processors. Each cell has the possibility of storing a large stack of binary images. The

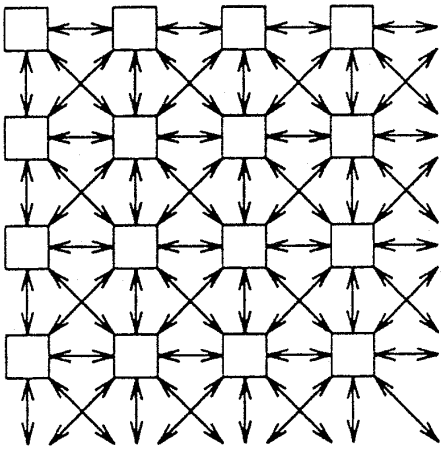

Fig. 5.2.10 Cellular array architecture

processing element can compute two independent binary outputs.

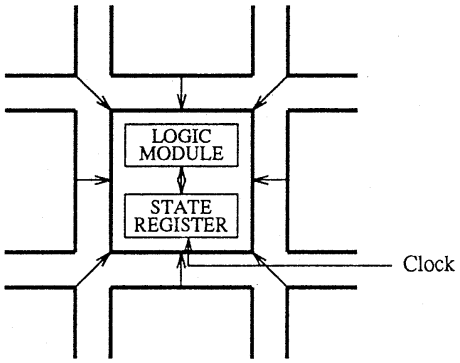

Fig. 5.2.11 Structure of a cell

One is transmitted to all neighbouring processors and the other corresponds to the new local image element. The computation time per image element is in the order of nanoseconds. CLIP can implement different programmable

neighbourhoods that have square or hexagonal shape. Several other cellular image processors exist e.g. Binary Image Processor (BIT), Massively Parallel Processor, Cytocomputer, etc.

Recently a new, parallel cellular array processor architecture the *cellular neural network, CNN* was proposed by [Chu88]. The basic architecture of CNN is similar to that what is shown in figure 5.2.10. CNN, however is a highly parallel analog array processor, which can be extended to form CNN universal machine [Ros92], a parallel hybrid computer array. In a CNN universal machine there is an analog CNN array processor some digital and analog registers for storing the data to be processed as well as some logic part to implement high-speed logic operations on the pixels. As CNN is a special neural network, its architecture and operation as well as, its main application fields will be presented in the next chapter.

Systolic array implementation of quadratic filters

In this section the systolic implementation of quadratic digital filter will be described. As it was written previously the input-output relationship of these filters can be given as:

$$y(n) = h_0 + \sum_{k=1}^{K} \bar{h}_k[x(n)], \tag{5.2.18}$$

where h_o is the constant term, K is the order of nonlinearity and $\bar{h}_k[x(n)]$ is defined as:

$$\bar{h}_k[x(n)] = \sum_{i_1=0}^{N-1} \ldots \sum_{i_k=0}^{N-1} h_k(i_1,\ldots,i_k)x(n-i_1)\ldots x(n-i_k) \tag{5.2.19}$$

Which has N^k terms. The linear term of the Volterra expansion is written as

$$y_1(n) = \bar{h}_1[x(n)] = \mathbf{X}_1^T(n)\mathbf{H}_1 \tag{5.2.20}$$

where

$$\mathbf{X}_1^T = [x(n)x(n-1)\ \ldots\ x(n-N+1)] \tag{5.2.21}$$

and

$$\mathbf{H}_1^T = [h_1(0)h_1(1)\ \ldots\ h_1(N-1)] \tag{5.2.22}$$

The general nonlinear term of order $k>1$ may be written in the form:

$$y_k(n) = \bar{h}_k[x(n)] = \mathbf{X}_k^T(n)\mathbf{H}_k \tag{5.2.23}$$

where

$$\mathbf{X}_k(n) = [\mathbf{X}_1(n) \otimes \mathbf{X}_{k-1}(n)] \tag{5.2.24}$$

and $\otimes$ denotes the Kronecker product of matrices.

It is easily seen that the quadratic term $y_2(n)$ may be written in the form:

$$y_2(n) = \bar{h}_2[x(n)] = \mathbf{X}_1^T(n)\mathbf{R}\mathbf{X}_1(n) \tag{5.2.25}$$

where **R** is a lower triangular matrix of the form.

$$\mathbf{R} = [r_{ij}] = \begin{bmatrix} h_2(0,0) & & & \\ h_2(0,1) & h_2(1,1) & & \\ \vdots & \vdots & & \\ h_2(0,N-1) & h_2(1,N-1) & \cdots & h_2(N-1,N-1) \end{bmatrix} \tag{5.2.26}$$

In the systolic implementation first the row-matrix multiplication $\mathbf{X}_1^T(n)\mathbf{R}$ will be implemented. In the sequel n more different PEs will be applied in the n outputs of the row-matrix multiplier. The detailed diagram of the proposed systolic array is shown in figure 5.2.12. Note that the structure suggested is nonrecursive, and therefore there are no feedbacks loops .

The execution time of the operations in each type of the used PEs equals to:

$$\tau = Mu + Ad \tag{5.2.27}$$

where Mu is the time needed by multiplication and Ad denotes the addition time. The throughput rate of the whole implememtation structure is given by:

$$R = \frac{1}{\tau} = \frac{1}{Mu + Ad} \tag{5.2.28}$$

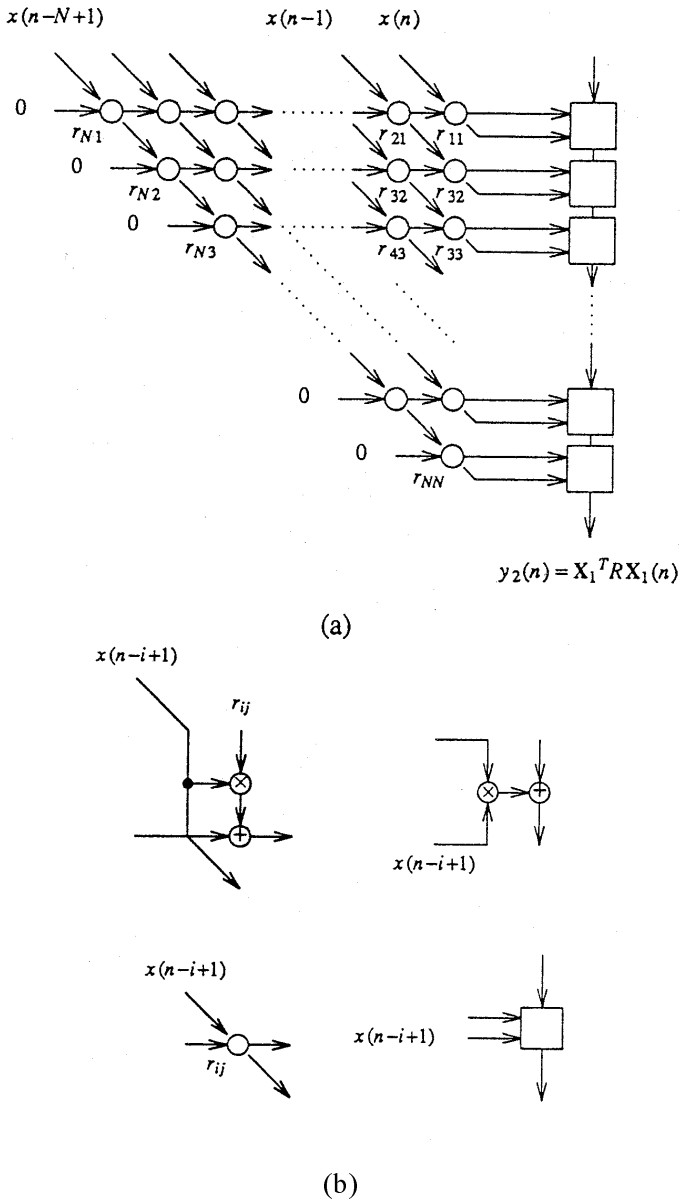

Fig. 5.2.12 Systolic array implementaion of the quadratic filter (a), the structure and the symbols of the used PEs (b)

Wavefront array implementation of quadratic filters

The SFG which is associated with the systolic array implementation, may be transformed to the data flow graph. The data flow graph of the wavefront array implementation is shown in figure 5.2.13.

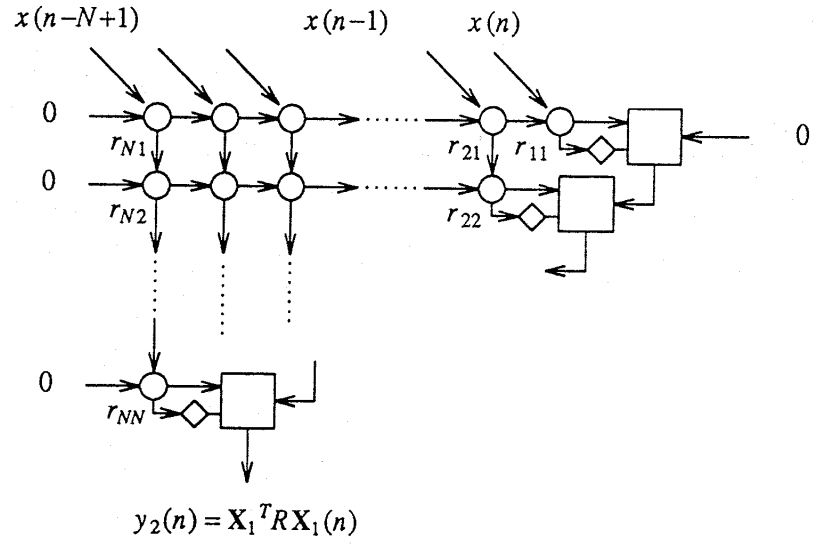

Fig. 5.2.13 Wavefront array implementation