Selesnick, I.W. & Burrus, C.S. “Fast Convolution and Filtering”
Digital Signal Processing Handbook
Ed. Vijay K. Madisetti and Douglas B. Williams
Boca Raton: CRC Press LLC, 1999

8

Fast Convolution and Filtering

Ivan W. Selesnick
Polytechnic University

C. Sidney Burrus
Rice University

8.1 Introduction

One of the first applications of the Cooley-Tukey fast Fourier transform (FFT) algorithm was to implement convolution faster than the usual direct method [13, 25, 30]. Finite impulse response (FIR) digital filters and convolution are defined by

$$y(n) = \sum_{k=0}^{L-1} h(k)\, x(n-k) \tag{8.1}$$

where, for an FIR filter, $x(n)$ is a length-N sequence of numbers considered to be the input signal, $h(n)$ is a length-L sequence of numbers considered to be the filter coefficients, and $y(n)$ is the filtered output. Examination of this equation shows that the output signal $y(n)$ must be a length-$(N+L-1)$ sequence of numbers, and the direct calculation of this output requires NL multiplications and approximately NL additions (actually, $(N-1)(L-1)$). If the signal and filter length are both length-N, we say the arithmetic complexity is of order N^2, $O(N^2)$. Our goal is calculate this convolution or filtering faster than directly implementing (8.1). The most common way to achieve "fast convolution" is to section or block the signal and use the FFT on these blocks to take advantage

of the efficiency of the FFT. Clearly, one disadvantage of this technique is an inherent delay of one block length.

Indeed, this approach is so common as to be almost synonymous with fast convolution. The problem is to implement on-going, noncyclic convolution with the finite-length, cyclic convolution that the FFT gives. An answer was quickly found in a clever organization of piecing together blocks of data using what is now called the *overlap-add* method and the *overlap-save* method. These two methods convolve length-L blocks using one length-L FFT, L complex multiplications, and one length-L inverse FFT [22].

Later this was generalized to arbitrary length blocks or sections to give block convolution and block recursion [5]. By allowing the block lengths to be even shorter than one word (bits and bytes!) we come up with an interesting implementation called distributed arithmetic that requires no explicit multiplications [7, 34].

Another approach for improving the efficiency of convolution and recursion uses fast algorithms other than the traditional FFT. One possibility is to use a transform based on number-theoretic roots of unity rather than the usual complex roots of unity [17]. This gives rise to number-theoretic transforms that require no multiplications and no trigonometric functions. Still another method applies Winograd's fast algorithms directly to convolution rather than through the Fourier transform. Finally, we remark that some filters $h(n)$ require fewer arithmetic operations because of their structure.

8.2 Overlap-Add and Overlap-Save Methods for Fast Convolution

If one implements convolution by use of the FFT, then it is cyclic convolution that is obtained. In order to use the FFT, zeros are appended to the signal or filter sequence until they are both the same length. If the FFT of the signal $x(n)$ is term-by-term multiplied by the FFT of the filter $h(n)$, the result is the FFT of the output $y(n)$. However, the length of $y(n)$ obtained by an inverse FFT is the same as the length of the input. Because the DFT or FFT is a periodic transform, the convolution implemented using this FFT approach is cyclic convolution, which means the output of (8.1) is wrapped or aliased. The tail of $y(n)$ is added to it head — but that is not usually what is wanted for filtering or normal convolution and correlation. This aliasing, the effects of cyclic convolution, can be overcome by appending zeros to both $x(n)$ and $h(n)$ until their lengths are $N+L-1$ and by then using the FFT. The part of the output that is aliased is zero and the result of the cyclic convolution is exactly the same as noncyclic convolution. The cost is taking the FFT of lengthened sequences — sequences for which about half the numbers are zero. Now that we can do noncyclic convolution with the FFT, how do we account for the effects of sectioning the input and output into blocks?

8.2.1 Overlap-Add

Because convolution is linear, the output of a long sequence can be calculated by simply summing the outputs of each block of the input. What is complicated is that the output blocks are longer than the input. This is dealt with by overlapping the tail of the output from the previous block with the beginning of the output from the present block. In other words, if the block length is N and it is greater than the filter length L, the output from the second block will overlap the tail of the output from the first block and they will simply be added. Hence the name: *overlap-add*. Figure 8.1 illustrates why the overlap-add method works, for $N = 10$, $L = 5$.

Combining the overlap-add organization with use of the FFT yields a very efficient algorithm for calculating convolution that is faster than direct calculation for lengths above 20 to 50. This cross-over point depends on the computer being used and the overhead needed by use of the FFTs.

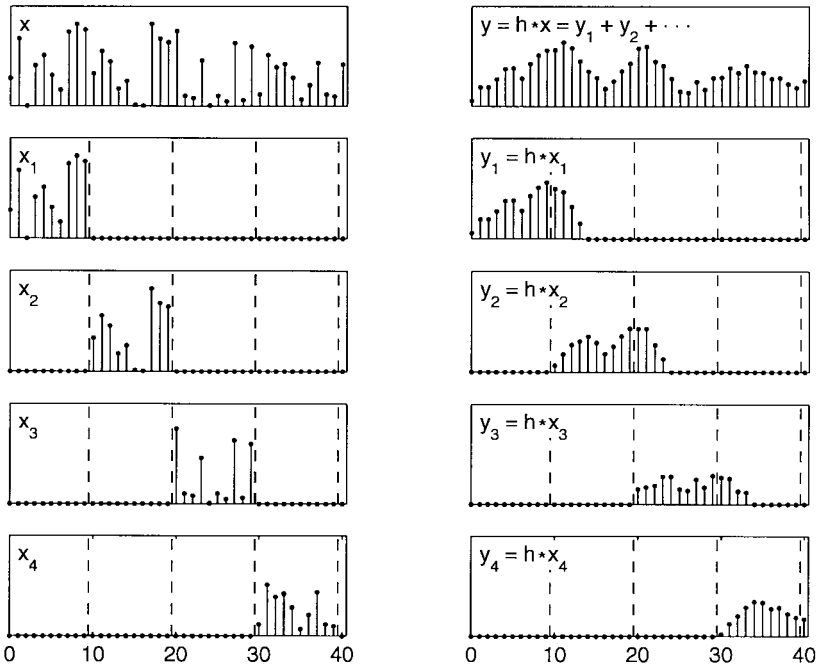

FIGURE 8.1: Overlap-add algorithm. The sequence $y(n)$ is the result of convolving $x(n)$ with an FIR filter $h(n)$ of length 5. In this example, $h(n) = 0.2$ for $n = 0, \ldots, 4$. The block length is 10, the overlap is 4. As illustrated in the figure, $x(n) = x_1(n) + x_2(n) + \cdots$ and $y(n) = y_1(n) + y_2(n) + \cdots$ where $y_i(n)$ is the result of convolving $x_i(n)$ with the filter $h(n)$.

8.2.2 Overlap-Save

A slightly different organization of the above approach is also often used for high-speed convolution. Rather than sectioning the input and then calculating the output from overlapped outputs from these individual input blocks, we will section the output and then use whatever part of the input contributes to that output block. In other words, to calculate the values in a particular output block, a section of length $N + L - 1$ from the input will be needed. The strategy is to save the part of the first input block that contributes to the second output block and use it in that calculation. It turns out that exactly the same amount of arithmetic and storage are used by these two approaches. Because it is the input that is now overlapped and, therefore, must be saved, this second approach is called *overlap-save*.

This method has also been called *overlap-discard* in [12] because, rather than adding the overlapping output blocks, the overlapping portion of the output blocks are discarded. As illustrated in Fig. 8.2, both the head and the tail of the output blocks are discarded. It may appear in Fig. 8.2 that an FFT of length 18 is needed. However, with the use of the FFT (to get cyclic convolution), the head and the tail overlap, so the FFT length is 14. (In practice, block lengths are generally chosen so that the FFT length $N + L - 1$ is a power of 2).

8.2.3 Use of the Overlap Methods

Because the efficiency of the FFT is $O(N \log(N))$, the efficiency of the overlap methods for convolution increases with length. To use the FFT for convolution will require one length-N forward FFT, N complex multiplications, and one length-N inverse FFT. The FFT of the filter is done once and

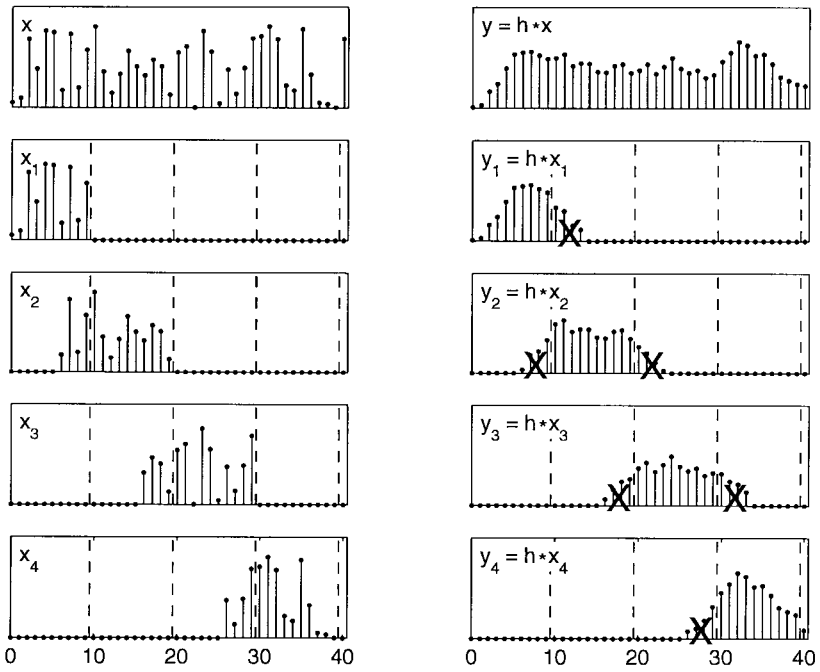

FIGURE 8.2: Overlap-save algorithm. The sequence $y(n)$ is the result of convolving $x(n)$ with an FIR filter $h(n)$ of length 5. In this example, $h(n) = 0.2$ for $n = 0, \ldots, 4$. The block length is 10, the overlap is 4. As illustrated in the figure, the sequence $y(n)$ is obtained, block by block, from the appropriate block of $y_i(n)$, where $y_i(n)$ is the result of convolving $x_i(n)$ with the filter $h(n)$.

stored rather than done repeatedly for each block. For short lengths, direct convolution will be more efficient. The exact length of filter where the efficiency cross-over occurs depends on the computer and software being used.

If it is determined that the FFT is potentially faster than direct convolution, the next question is what block length to use. Here, there is a compromise between the improved efficiency of long FFTs and the fact you are processing a lot of appended zeros that contribute nothing to the output. An empirical plot of multiplication (and, perhaps, additions) per output point vs. block length will have a minimum that may be several times the filter length. This is an important parameter that should be optimized for each implementation. Remember that this increased block length may improve efficiency but it adds a delay and requires memory for storage.

8.3 Block Convolution

The operation of a finite impulse response (FIR) filter is described by a finite convolution as

$$y(n) = \sum_{k=0}^{L-1} h(k)\, x(n-k) \tag{8.2}$$

where $x(n)$ is causal, $h(n)$ is causal and of length L, and the time index n goes from zero to infinity or some large value. With a change of index variables this becomes

$$y(n) = \sum_{k=0}^{n} h(n-k)\, x(k) \tag{8.3}$$

which can be expressed as a matrix operation by

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & \cdots & 0 \\ h_1 & h_0 & 0 & & \\ h_2 & h_1 & h_0 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix}. \tag{8.4}$$

The H matrix of impulse response values is partitioned into N by N square submatrices and the X and Y vectors are partitioned into length-N blocks or sections. This is illustrated for $N = 3$ by

$$H_0 = \begin{bmatrix} h_0 & 0 & 0 \\ h_1 & h_0 & 0 \\ h_2 & h_1 & h_0 \end{bmatrix}, \qquad H_1 = \begin{bmatrix} h_3 & h_2 & h_1 \\ h_4 & h_3 & h_2 \\ h_5 & h_4 & h_3 \end{bmatrix}, \qquad \text{etc.} \tag{8.5}$$

$$\underline{x}_0 = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}, \qquad \underline{x}_1 = \begin{bmatrix} x_3 \\ x_4 \\ x_5 \end{bmatrix}, \qquad \underline{y}_0 = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}, \qquad \text{etc.} \tag{8.6}$$

Substituting these definitions into (8.4) gives

$$\begin{bmatrix} \underline{y}_0 \\ \underline{y}_1 \\ \underline{y}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} H_0 & 0 & 0 & \cdots & 0 \\ H_1 & H_0 & 0 & & \\ H_2 & H_1 & H_0 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \underline{x}_2 \\ \vdots \end{bmatrix} \tag{8.7}$$

The general expression for the n^{th} output block is

$$\underline{y}_n = \sum_{k=0}^{n} H_{n-k}\, \underline{x}_k \tag{8.8}$$

which is a vector or block convolution. Since the matrix-vector multiplication within the block convolution is itself a convolution, (8.9) is a sort of convolution of convolutions and the finite length matrix-vector multiplication can be carried out using the FFT or other fast convolution methods.

The equation for one output block can be written as the product

$$\underline{y}_2 = \begin{bmatrix} H_2 & H_1 & H_0 \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} \tag{8.9}$$

and the effects of one input block can be written

$$\begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} \underline{x}_1 = \begin{bmatrix} \underline{y}_0 \\ \underline{y}_1 \\ \underline{y}_2 \end{bmatrix}. \tag{8.10}$$

These are generalized statements of overlap-save and overlap-add [11, 30]. The block length can be longer, shorter, or equal to the filter length.

8.3.1 Block Recursion

Although less well known, infinite impulse response (IIR) filters can be implemented with block processing [5, 6]. The block form of an IIR filter is developed in much the same way as the block convolution implementation of the FIR filter. The general constant coefficient difference equation which describes an IIR filter with recursive coefficients a_l, convolution coefficients b_k, input signal $x(n)$, and output signal $y(n)$ is given by

$$y(n) = \sum_{l=1}^{N-1} a_l \, y_{n-l} + \sum_{k=0}^{M-1} b_k \, x_{n-k} \tag{8.11}$$

using both functional notation and subscripts, depending on which is easier and clearer. The impulse response $h(n)$ is

$$h(n) = \sum_{l=1}^{N-1} a_l \, h(n-l) + \sum_{k=0}^{M-1} b_k \, \delta(n-k) \tag{8.12}$$

which, for $N = 4$, can be written in matrix operator form

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_1 & 1 & 0 & & \\ a_2 & a_1 & 1 & & \\ a_3 & a_2 & a_1 & & \\ 0 & a_3 & a_2 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ 0 \\ \vdots \end{bmatrix}$$

In terms of smaller submatrices and blocks, this becomes

$$\begin{bmatrix} A_0 & 0 & 0 & \cdots & 0 \\ A_1 & A_0 & 0 & & \\ 0 & A_1 & A_0 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} \underline{h}_0 \\ \underline{h}_1 \\ \underline{h}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \underline{b}_0 \\ \underline{b}_1 \\ 0 \\ \vdots \end{bmatrix} \tag{8.13}$$

for blocks of dimension two. From this formulation, a block recursive equation can be written that will generate the impulse response block by block.

$$A_0 \, \underline{h}_n + A_1 \, \underline{h}_{n-1} = 0 \qquad \text{for } n \geq 2 \tag{8.14}$$

or

$$\underline{h}_n = -A_0^{-1} A_1 \, \underline{h}_{n-1} = K \, \underline{h}_{n-1} \qquad \text{for } n \geq 2 \tag{8.15}$$

with initial conditions given by

$$\underline{h}_1 = -A_0^{-1} A_1 A_0^{-1} \, \underline{b}_0 + A_0^{-1} \, \underline{b}_1 \tag{8.16}$$

Next, we develop the recursive formulation for a general input as described by the scalar difference equation (8.12) and in matrix operator form by

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ a_1 & 1 & 0 & & \\ a_2 & a_1 & 1 & & \\ a_3 & a_2 & a_1 & & \\ 0 & a_3 & a_2 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_0 & 0 & 0 & \cdots & 0 \\ b_1 & b_0 & 0 & & \\ b_2 & b_1 & b_0 & & \\ 0 & b_2 & b_1 & & \\ 0 & 0 & b_2 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{bmatrix} \tag{8.17}$$

which, after substituting the definitions of the submatrices and assuming the block length is larger than the order of the numerator or denominator, becomes

$$\begin{bmatrix} A_0 & 0 & 0 & \cdots & 0 \\ A_1 & A_0 & 0 & & \\ 0 & A_1 & A_0 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} \underline{y}_0 \\ \underline{y}_1 \\ \underline{y}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} B_0 & 0 & 0 & \cdots & 0 \\ B_1 & B_0 & 0 & & \\ 0 & B_1 & B_0 & & \\ \vdots & & & & \vdots \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \underline{x}_2 \\ \vdots \end{bmatrix} . \tag{8.18}$$

From the partitioned rows of (8.19), one can write the block recursive relation

$$A_0\, \underline{y}_{n+1} + A_1\, \underline{y}_n = B_0\, \underline{x}_{n+1} + B_1\, \underline{x}_n \tag{8.19}$$

Solving for $\underline{y}_{n+1}$ gives

$$\underline{y}_{n+1} = -A_0^{-1} A_1\, \underline{y}_n + A_0^{-1} B_0\, \underline{x}_{n+1} + A_0^{-1} B_1\, \underline{x}_n \tag{8.20}$$

$$\underline{y}_{n+1} = K\, \underline{y}_n + H_0\, \underline{x}_{n+1} + \tilde{H}_1\, \underline{x}_n \tag{8.21}$$

which is a first order vector difference equation [5, 6]. This is the fundamental block recursive algorithm that implements the original scalar difference equation in (8.12). It has several important characteristics.

1. The block recursive formulation is similar to a state variable equation but the states are blocks or sections of the output [6].
2. If the block length were shorter than the denominator, the vector difference equation would be higher than first order. There would be a nonzero A_2. If the block length were shorter than the numerator, there would be a nonzero B_2 and a higher order block convolution operation. If the block length were one, the order of the vector equation would be the same as the scalar equation. They would be the same equation.
3. The actual arithmetic that goes into the calculation of the output is partly recursive and partly convolution. The longer the block, the more the output is calculated by convolution, and the more arithmetic is required.
4. There are several ways of using the FFT in the calculation of the various matrix products in (8.20). Each has some arithmetic advantage for various forms and orders of the original equation. It is also possible to implement some of the operations using rectangular transforms, number theoretic transforms, distributed arithmetic, or other efficient convolution algorithms [6, 36].

8.4 Short and Medium Length Convolution

For the cyclic convolution of short sequences ($n \leq 10$) and medium length sequences ($n \leq 100$), special algorithms are available. For short lengths, algorithms that require the minimum number of multiplications possible have been developed by Winograd [8, 17, 35]. However, for longer lengths Winograd's algorithms, based on his theory of multiplicative complexity, require a large number of additions and become cumbersome to implement. Nesting algorithms, such as the Agarwal-Cooley and split-nesting algorithm, are methods that combine short convolutions. By nesting Winograd's short convolution algorithms, efficient medium length convolution algorithms can thereby be obtained.

In the following section we give a matrix description of these algorithms and of the Toom-Cook algorithm. Descriptions based on polynomials can be found in [4, 8, 19, 21, 24]. The presentation that

follows relies upon the notions of similarity transformations, companion matrices, and Kronecker products. With them, the algorithms are described in a manner that brings out their structure and differences. It is found that when companion matrices are used to describe cyclic convolution, the algorithms block-diagonalize the cyclic shift matrix.

8.4.1 The Toom-Cook Method

A basic technique in fast algorithms for convolution is interpolation: two polynomials are evaluated at some common points, these values are multiplied, and by computing the polynomial interpolating these products, the product of the two original polynomials is determined [4, 19, 21, 31]. This interpolation method is often called the Toom-Cook method and can be described by a bilinear form. Let $n = 2$,

$$\begin{aligned} X(s) &= x_0 + x_1 s + x_2 s^2 \\ H(s) &= h_0 + h_1 s + h_2 s^2 \\ Y(s) &= y_0 + y_1 s + y_2 s^2 + y_3 s^3 + y_4 s^4 \,. \end{aligned}$$

The linear convolution of x and h can be represented by a matrix-vector product $y = Hx$,

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} h_0 & & \\ h_1 & h_0 & \\ h_2 & h_1 & h_0 \\ & h_2 & h_1 \\ & & h_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

or as a polynomial product $Y(s) = H(s)X(s)$. In the former case, the linear convolution matrix can be written as $h_0 H_0 + h_1 H_1 + h_2 H_2$ where the meaning of H_k is clear. In the later case, one obtains the expression

$$y = C\,\{Ah * Ax\} \tag{8.22}$$

where $*$ denotes point-by-point multiplication. The terms Ah and Ax are the values of $H(s)$ and $X(s)$ at some points $i_1, \ldots i_{2n-1}$ ($n = 2$). The point-by-point multiplication gives the values $Y(i_1), \ldots, Y(i_{2n-1})$. The operation of C obtains the coefficients of $Y(s)$ from its values at the point $i_1, \ldots i_{2n-1}$. Equation (8.22) is a bilinear form and it implies that

$$H_k = C \text{ diag } (Ae_k)A$$

where e_k is the kth standard basis vector. (Ae_k is the kth column of A). However, A and C do not need to be Vandermonde matrices as suggested above. As long as A and C are matrices such that $H_k = C$ diag $(Ae_k)A$, then the linear convolution of x and h is given by the bilinear form $y = C\{Ah * Ax\}$. More generally, as long as A, B, and C are matrices satisfying $H_k = C$ diag $(Be_k)A$, then $y = C\{Bh * Ax\}$ computes the linear convolution of h and x. For convenience, if $C\{Bh * Ax\}$ computes the n point linear convolution of h and x (both h and x are n point sequences), then we say "(A, B, C) describes a bilinear form for n point linear convolution."

EXAMPLE 8.1:

(A, A, C) describes a 2-point linear convolution where

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}. \tag{8.23}$$

8.4.2 Cyclic Convolution

The cyclic convolution of x and h can be represented by a matrix-vector product

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_0 & h_2 & h_1 \\ h_1 & h_0 & h_2 \\ h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

or as the remainder of a polynomial product after division by s^n-1, denoted by $Y(s) = \langle H(s)X(s)\rangle_{s^n-1}$. In the former case, the cyclic convolution matrix can be written as $h_0 I + h_1 S_2 + h_2 S_2^2$ where S_n is the cyclic shift matrix,

$$S_n = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}.$$

It will be useful to make a more general statement.

The companion matrix of a monic polynomial, $M(s) = m_0 + m_1 s + \cdots + m_{n-1}s^{n-1} + s^n$ is given by

$$C_M = \begin{bmatrix} & & & -m_0 \\ 1 & & & -m_1 \\ & \ddots & & \vdots \\ & & 1 & -m_{n-1} \end{bmatrix}.$$

Its usefulness in the following discussion comes from the following relation, which permits a matrix formulation of convolution:

$$Y(s) = \langle H(s)X(s)\rangle_{M(s)} \quad \Longleftrightarrow \quad y = \left(\sum_{k=0}^{n-1} h_k C_M^k\right) x \tag{8.24}$$

where x, h, and y are the vectors of coefficients and C_M is the companion matrix of $M(s)$. In (8.24), we say y is the convolution of x and h with respect to $M(s)$. In the case of cyclic convolution, $M(s) = s^n - 1$ and C_{s^n-1} is the cyclic shift matrix, S_n.

Similarity transformations can be used to interpret the action of some convolution algorithms. If $C_M = T^{-1}QT$ for some matrix T (C_M and Q are similar, denoted $C_M \sim Q$), then (8.24) becomes

$$y = T^{-1}\left(\sum_{k=0}^{n-1} h_k Q^k\right) Tx\ .$$

That is, by employing the similarity transformation given by T in this way, the action of S_n^k is replaced by that of Q^k. Many cyclic convolution algorithms can be understood, in part, by understanding the manipulations made to S_n and the resulting new matrix Q. If the transformation T is to be useful, it must satisfy two requirements: (1) Tx must be simple to compute, and (2) Q must have some advantageous structure. For example, by the convolution property of the DFT, the DFT matrix F diagonalizes S_n and, therefore, it diagonalizes every circulant matrix. In this case, Tx can be computed by an FFT and the structure of Q is the simplest possible: a diagonal.

8.4.3 Winograd Short Convolution Algorithm

The Winograd algorithm [35] can be described using the notation above. Suppose $M(s)$ can be factored as $M(s) = M_1(s)M_2(s)$ where $M_1(s)$ and $M_2(s)$ have no common roots, then $C_M \sim$

$\left(C_{M_1} \oplus C_{M_2}\right)$ where $\oplus$ denotes the matrix direct sum. Using this similarity and recalling (8.24), the original convolution can be decomposed into two disjoint convolutions. This is a statement of the Chinese remainder theorem for polynomials expressed in matrix notation. In the case of cyclic convolution, $s^n - 1$ can be written as the product of *cyclotomic* polynomials — polynomials whose coefficients are small integers. Denoting the dth cyclotomic polynomial by $\Phi_d(s)$, one has $s^n - 1 = \prod_{d|n} \Phi_d(s)$. Therefore, S_n can be transformed to a block diagonal matrix,

$$S_n \sim \begin{bmatrix} C_{\Phi_1} & & & \\ & C_{\Phi_d} & & \\ & & \ddots & \\ & & & C_{\Phi_n} \end{bmatrix} = \left(\bigoplus_{d|n} C_{\Phi_d} \right) . \tag{8.25}$$

The symbol $\oplus$ denotes the matrix direct sum (diagonal concatenation). Each matrix on the diagonal is the companion matrix of a cyclotomic polynomial.

EXAMPLE 8.2:

$$\begin{aligned} s^{15} - 1 &= \Phi_1(s)\Phi_3(s)\Phi_5(s)\Phi_{15}(s) \\ &= (s-1)(s^2+s+1)(s^4+s^3+s^2+s+1)(s^8-s^7+s^5-s^4+s^3-s+1) \end{aligned}$$

$$S_{15} = T^{-1} \begin{bmatrix} 1 & & & & & & & & & & & & & & \\ & -1 & & & & & & & & & & & & & \\ 1 & -1 & & & & & & & & & & & & & \\ & & & & -1 & & & & & & & & & & \\ & & 1 & & -1 & & & & & & & & & & \\ & & & 1 & -1 & & & & & & & & & & \\ & & & & -1 & & & & & & & & & & \\ & & & & & & & & & & & & -1 & & \\ & & & & & & 1 & & & & & & 1 & & \\ & & & & & & & 1 & & & & & & & \\ & & & & & & & & 1 & & & & -1 & & \\ & & & & & & & & & 1 & & & 1 & & \\ & & & & & & & & & & 1 & & -1 & & \\ & & & & & & & & & & & 1 & & & \\ & & & & & & & & & & & & 1 & 1 & \end{bmatrix} T . \tag{8.26}$$

Each block represents a convolution with respect to a cyclotomic polynomial, or a "cyclotomic convolution." When n has several prime divisors the similarity transformation T becomes quite complicated. However, when n is a prime power, the transformation is very structured, as described in [29].

As in the previous section, we can write a bilinear form for cyclotomic convolution. Let d be any positive integer and let $X(s)$ and $H(s)$ be polynomials of degree $\phi(d)-1$ where $\phi(\cdot)$ is the Euler totient function. If A, B, and C are matrices satisfying $\left(C_{\Phi_d}\right)^k = C \text{ diag } (Be_k)A$ for $0 \le k \le \phi(d) - 1$, then the coefficients of $Y(s) = \langle X(s)H(s) \rangle_{\Phi_d(s)}$ are given by $y = C\{Bh * Ax\}$. As above, for such A, B, and C, we say "(A, B, C) describes a bilinear form for $\Phi_d(s)$ convolution."

But since $\langle X(s)H(s) \rangle_{\Phi_d(s)}$ can be found by computing the product of $X(s)$ and $H(s)$ and reducing the result, a cyclotomic convolution algorithm can always be derived by following a linear convolution algorithm by the appropriate reduction operation: If G is the appropriate reduction matrix and if (A, B, C) describes a bilinear form for a $\phi(d)$ point *linear* convolution, then (A, B, GC) describes a bilinear form for $\Phi_d(s)$ convolution. That is, $y = GC\{Bh * Ax\}$ computes the coefficients of $\langle X(s)H(s) \rangle_{\Phi_d(s)}$.

EXAMPLE 8.3:

A bilinear form for $\Phi_3(s)$ convolution is described by $(A,\ A,\ GC)$ where A and C are given in (8.23) and G is given by

$$G = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} .$$

The Winograd short cyclic convolution algorithm decomposes the convolution into smaller (cyclotomic) ones, and can be described as follows. If (A_d, B_d, C_d) describes a bilinear form for $\Phi_d(s)$ convolution, then a bilinear form for cyclic convolution is provided by

$$A = \left(\oplus_{d|n} A_d\right) T \qquad B = \left(\oplus_{d|n} B_d\right) T \qquad C = T^{-1} \left(\oplus_{d|n} C_d\right) .$$

The matrix T decomposes the problem into disjoint parts, and T^{-1} recombines the results.

8.4.4 The Agarwal-Cooley Algorithm

The Agarwal-Cooley [3] algorithm uses a similarity of another form. Namely, when $n = n_1 n_2$, and $(n_1, n_2) = 1$

$$S_n = P^t \left(S_{n_1} \otimes S_{n_2}\right) P \tag{8.27}$$

where $\otimes$ denotes the Kronecker product and P is a permutation matrix. The permutation is $k \to \langle k \rangle_{n_1} + n_1 \langle k \rangle_{n_2}$. This converts a one-dimensional cyclic convolution of length n into a two-dimensional one of length n_1 along one dimension and length n_2 along the second. Then an n_1-point and an n_2-point cyclic convolution algorithm can be combined to obtain an n-point algorithm.

8.4.5 The Split-Nesting Algorithm

The split-nesting algorithm [21] combines the structures of the Winograd and Agarwal-Cooley methods, so that S_n is transformed to a block diagonal matrix as in (8.25),

$$S_n \sim \bigoplus_{d|n} \Psi(d) . \tag{8.28}$$

Here $\Psi(d) = \bigotimes_{p|d,\, p\in\mathcal{P}} C_{\Phi_{H_d(p)}}$ where $H_d(p)$ is the highest power of p dividing d, and $\mathcal{P}$ is the set of primes. An example clarifies this decomposition.

EXAMPLE 8.4:

$$S_{45} = P^t R^{-1} \begin{bmatrix} 1 & & & & & & \\ & C_{\Phi_3} & & & & & \\ & & C_{\Phi_9} & & & & \\ & & & C_{\Phi_5} & & & \\ & & & & C_{\Phi_3} \otimes C_{\Phi_5} & & \\ & & & & & C_{\Phi_9} \otimes C_{\Phi_5} & \end{bmatrix} RP \tag{8.29}$$

where P is the same permutation matrix of (8.27), and R is a matrix described in [29].

In the split-nesting algorithm, each matrix along the diagonal represents a multidimensional cyclotomic convolution rather than a one-dimensional one. To obtain a bilinear form for the split-nesting method, bilinear forms for one-dimensional convolutions can be combined to obtain bilinear forms for multi-dimensional cyclotomic convolution. This is readily explained by an example.

EXAMPLE 8.5:

A 45-point circular convolution algorithm:

$$y = P^t R^{-1} C \{BRPh * ARPx\} \tag{8.30}$$

where

$$\begin{aligned} A &= 1 \oplus A_3 \oplus A_9 \oplus A_5 \oplus (A_3 \otimes A_5) \oplus (A_9 \otimes A_5) \\ B &= 1 \oplus B_3 \oplus B_9 \oplus B_5 \oplus (B_3 \otimes B_5) \oplus (B_9 \otimes B_5) \\ C &= 1 \oplus C_3 \oplus C_9 \oplus C_5 \oplus (C_3 \otimes C_5) \oplus (C_9 \otimes C_5) \end{aligned}$$

and where $(A_{p^i}, B_{p^i}, C_{p^i})$ describes a bilinear form for $\Phi_{p^i}(s)$ convolution.

Split-nesting (1) requires a simpler similarity transformation than the Winograd algorithm and (2) decomposes cyclic convolution into several disjoint multidimensional convolutions. For these reasons, for medium lengths, split-nesting can be more efficient than the Winograd convolution algorithm, even though it does not achieve the minimum number of multiplications. An explicit matrix description of the similarity transformation is provided in [29].

8.5 Multirate Methods for Running Convolution

While fast FIR filtering, based on block processing and the FFT, is computationally efficient, for real-time processing it has three drawbacks: (1) A delay is incurred; (2) the multiply-accumulate structure of the convolutional sum, a command for which DSPs are optimized, is lost; and (3) extra memory and communication (data transfer) time is needed. For real-time applications, this has motivated the development of alternative methods for convolution that partially retain the FIR filtering structure [18, 33].

In the z-domain, the running convolution of x and h is described by a polynomial product

$$Y(z) = H(z)X(z) \tag{8.31}$$

where $X(z)$ and $Y(z)$ are of infinite degree, and $H(z)$ is of finite degree. Let us write the polynomials as follows

$$X(z) = X_0\left(z^2\right) + z^{-1}X_1\left(z^2\right) \tag{8.32}$$

$$Y(z) = Y_0\left(z^2\right) + z^{-1}Y_1\left(z^2\right) \tag{8.33}$$

$$H(z) = H_0\left(z^2\right) + z^{-1}H_1\left(z^2\right) \tag{8.34}$$

where

$$X_0(z) = \sum_{i=0}^{\infty} x_{2i} z^{-i} \qquad X_1(z) = \sum_{i=0}^{\infty} x_{2i+1} z^{-i}$$

and Y_0, Y_1, H_0, H_1 are similarly defined. (These are known as polyphase components, although that is not important here). The polynomial product (8.31) can then be written as

$$Y_0\left(z^2\right) + z^{-1}Y_1\left(z^2\right) = \left(H_0\left(z^2\right) + z^{-1}H_1\left(z^2\right)\right)\left(X_0\left(z^2\right) + z^{-1}X_1\left(z^2\right)\right) \tag{8.35}$$

or in matrix form as

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-2}H_1 \\ H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \tag{8.36}$$

where $Y_0 = Y_0(z^2)$, etc.

The general form of (8.34) is given by

$$X(z) = \sum_{k=0}^{N-1} z^{-1} X_k(z^N)$$

where

$$X_k(z) = \sum_i x_{Ni+k} z^{-i}$$

and similarly for H and Y. For clarity, $N = 2$ is used in this exposition.

Note that the right hand side of (8.35) is a product of two polynomials of degree N, where the coefficients are themselves polynomials, either of finite degree (H_i), or of infinite degree (X_i). Accordingly, the Toom-Cook algorithm described previously can be employed, in which case the sums and products become polynomial sums and products. The essential key is that the polynomial products are themselves equivalent to FIR filtering, with shorter filters.

A Toom-Cook algorithm for carrying out (8.35) is given by

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = C \left\{ A \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} * A \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \right\}$$

where

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{bmatrix}.$$

This Toom-Cook algorithm yields the multirate filter bank structure shown in Fig. 8.3. The outputs of the two downsamplers, on the left side of the structure shown in the figure, are $X_0(z)$ and $X_1(z)$. The outputs of the two upsamplers, on the right side of the structure, are $Y_0(z^2)$ and $Y_1(z^2)$. Note that the three filters H_0, $H_0 + H_1$, and H_1 operate at half the sampling rate. The right-most operation shown in Fig. 8.3 is not an arithmetic addition — it is a merging of the two sequences, $Y_0(z^2)$ and $z^{-1}Y_1(z^2)$, by interleaving. The arithmetic overhead is 1 "input" addition and 3 "output" additions per 2 samples; that is a total of 2 additions per sample.

If the original filter $H(z)$ is of length L and operates at the rate f_s, then the structure in Fig. 8.3 is an implementation of $H(z)$ that employs three filters of length $L/2$, each operating at the rate $\frac{1}{2} f_s$.

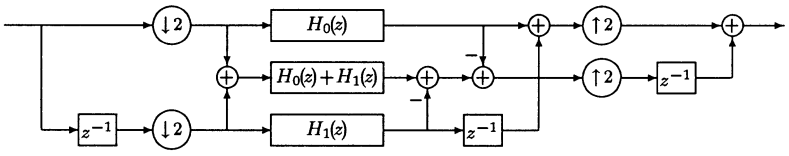

FIGURE 8.3: Filter structure based on a two-point convolution algorithm. Let H_0 be the even coefficients of a filter H, let H_1 be the odd coefficients. The structure implements the filter H using three half-length filters, each running at half the rate of H.

The convolutional sum for $H(z)$, when implemented directly, requires L multiplications per output point and $L-1$ additions per output point. Per output point, the structure in Fig. 8.3 requires $\frac{3}{4}L$ multiplications and $2 + \frac{3}{2}(L/2 - 1) = \frac{3}{4}L + \frac{1}{2}$ additions.

The decomposition can be repeatedly applied to each of the three filters; however, the benefit diminishes for small L, and quantization errors may accumulate. Table 8.1 gives the number of multiplications needed to implement a length 32 FIR filter, using various levels of decomposition.

TABLE 8.1 Computation of Running Convolution

Method	Subsampling	Delay	Mult./Point
1 32-pt. FIR filter	1	0	32
3 16-pt. FIR filters	2	1	24
9 8-pt. FIR filters	4	3	18
27 4-pt. FIR filters	8	7	13.5
81 2-pt. FIR filters	16	15	10.125
243 1-pt. mults.	32	31	7.59

Based on repeated application of two-point convolution structure in Fig. 8.3. (From [33].)

Other short linear convolution algorithms can be obtained from existing ones by a technique known as transposition. The transposed form of a short convolution algorithm has the same arithmetic complexity, but in a different arrangement. It was observed in [18] that the transposed forms generally have more input additions and fewer output additions. Consequently, the transposed forms should be more robust to quantization noise.

Various short-length convolution algorithms that are appropriate for this approach are provided in [18]. Also addressed is the issue of when to stop successive decompositions — and the problem of finding the best way to combine small-length filters, depending on various criteria. In particular, it is noted that DSPs generally perform a multiply-accumulate (MAC) operation in a single clock cycle, in which case a MAC should be considered a single operation.

It appears that this approach is amenable to (1) efficient multiprocessor implementations due to their inherent parallelism, and (2) efficient VLSI realization, since the implementation requires only local communication, instead of global exchange of data as in the case of FFT-based algorithms.

In [33], the following is noted. The mapping of long convolutions into small, subsampled convolutions is attractive in hardware (VLSI), software (signal processors), and multiprocessor implementations since the basic building blocks remain convolutions which can be computed efficiently once small enough.

8.6 Convolution in Subbands

Maximally decimated perfect reconstruction filter banks have been used for a variety of applications where processing in subbands is advantageous. Such filter banks can be regarded as generalizations of the short-time Fourier transform, and it turns out that the convolution theorem can be extended to them [23, 32]. In other words, the convolution of two signals can be found by directly convolving the subband signals and combining the results. In [23], both uniform and nonuniform decimation ratios are considered for orthonormal and biorthonormal filter banks. In [32], the results of [23] are generalized.

The advantage of this method is that the subband signals can be quantized based on the signal variance in each subband and other perceptual considerations, as in traditional subband coding. Instead of quantizing $x(n)$ and then convolving with $g(n)$, the subbands $x_k(n)$ and $g_k(n)$ are quantized, and the results are added. When quantizing in the subbands, the subband energy distribution can be exploited and bits can be allocated to subbands accordingly. For a fixed bit rate, this approach increases the accuracy of the overall convolution — that is, this approach offers a coding gain.

In [23] an optimal bit allocation formula and the optimized coding gain is derived for orthogonal filter banks. The contribution to coding gain comes partly from the nonuniformity of the signal

spectrum and partly from the nonuniformity of the filter spectrum. When the filter impulse response is taken to be the unit impulse $\delta(n)$, the formulas for the bit allocation and coding gain reduce to those for traditional subband and transform coding.

The efficiency that is gained from subband convolution comes from the ability to use a fewer number of bits to achieve a given level of accuracy. In addition, in [23], low sensitivity filter structures are derived from the subband convolution theorem and examined.

8.7 Distributed Arithmetic

Rather than grouping the individual scalar data values in a discrete-time signal into blocks, the scalar values can be partitioned into groups of bits. Because multiplication of integers, multiplication of polynomials, and discrete-time convolution are the same operations, the bit-level description of multiplication can be mixed with the convolution of the signal processing. The resulting structure is called distributed arithmetic [7, 34].

8.7.1 Multiplication is Convolution

To simplify the presentation, we will assume the data and coefficients to be positive integers with simple binary coding and the problem of carrying will be omitted. Assume the product of two B-bit words is desired

$$y = a\,x \tag{8.37}$$

where

$$a = \sum_{i=0}^{B-1} a_i 2^i \quad \text{and} \quad x = \sum_{i=0}^{B-1} a_j 2^j \tag{8.38}$$

with $a_i, x_j \in \{0, 1\}$. This gives

$$y = \sum_i a_i 2^i \sum_j x_j 2^j \tag{8.39}$$

which, with a change of variables $k = i + j$, becomes

$$y = \sum_k \sum_i a_i\, x_{k-i} 2^k \,. \tag{8.40}$$

Using the binary description of y as

$$y = \sum_k y_k 2^k \tag{8.41}$$

we have for the binary coefficients

$$y_k = \sum_i a_i\, x_{k-i} \tag{8.42}$$

as a convolution of the binary coefficients for a and x. We see that multiplying two numbers is the same as convolving their coefficient representation any base. Multiplication is convolution.

8.7.2 Convolution is Two Dimensional

Consider the following convolution of number strings (FIR filtering)

$$y(n) = \sum_\ell a(\ell)\, x(n - \ell) \,. \tag{8.43}$$

Using the binary representation of the coefficients and data, we have

$$y(n) = \sum_{\ell} \sum_{i} a_i(\ell)\, 2^i \sum_{j} x_j(n-\ell)\, 2^j \tag{8.44}$$

$$y(n) = \sum_{\ell} \sum_{i} \sum_{i} a_i(\ell) x_j(n-\ell) 2^{i+j} \tag{8.45}$$

which after changing variables, $k = i + j$, becomes

$$y(n) = \sum_{k} \sum_{\ell} \sum_{i} a_i(\ell)\, x_{k-i}(n-\ell)\, 2^k \,. \tag{8.46}$$

A one-dimensional convolution of numbers is a two-dimensional convolution of the binary (or other base) representations of the numbers.

8.7.3 Distributed Arithmetic by Table Lookup

The usual way that distributed arithmetic convolution is calculated does the arithmetic in a special concentrated algorithm or piece of hardware. We are now going to reorder the very general description in (8.46) to allow some of the operations to be precomputed and stored in a lookup table. The arithmetic will then be distributed with the convolution itself.

If (8.46) is summed over the index i, we have

$$y(n) = \sum_{j} \sum_{\ell} a(\ell)\, x_j(n-\ell)\, 2^j \,. \tag{8.47}$$

Each sum over ℓ convolves the word string $a(n)$ with the bit string $x_j(n)$ to produce a partial product which is then shifted and added by the sum over j to give $y(n)$.

If (8.47) is summed over ℓ to form a table which can be addressed by the binary numbers $x_j(n)$, we have

$$y(n) = \sum_{j} f(x_j(n), x_j(n-1), \cdots)\, 2^j \tag{8.48}$$

where

$$f(x_j(n), x_j(n-1), \cdots) = \sum_{\ell} a(\ell)\, x_j(n-\ell) \tag{8.49}$$

The numbers $a(i)$ are the coefficients of the filter, which as usual is assumed to be fixed. Consider a filter of length L. This function $f()$ is a function of L binary variables and, therefore, takes on 2^L possible values. The function is determined by the filter, $a(i)$. For example, if $L = 3$, the table (function values) would contain eight values:

$$0,\ a(0),\ a(1),\ a(2),\ (a(0)+a(1)),\ (a(1)+a(2)),\ (a(0)+a(2)),\ (a(0)+a(1)+a(2)) \tag{8.50}$$

and if the words were stored as B bits, they would require $2^L B$ bits of memory.

There are extensions and modifications of this basic idea to allow a very flexible trade of memory for logic. The idea is to precompute as much as possible, store it in a table, and fetch it when needed. The two extremes of this are on one hand to compute all possible outputs and simply fetch them using the input as an address. The other extreme is the usual system which simply stores the coefficients and computes what is needed as needed.

This table lookup is illustrated in Fig. 8.4 where the blocks represent 4 b words, where the least significant bit of each of the four most recent data words form the address for the table lookup from memory. After 4 b shifts and accumulates, the output word $y(n)$ is available, using no multiplications.

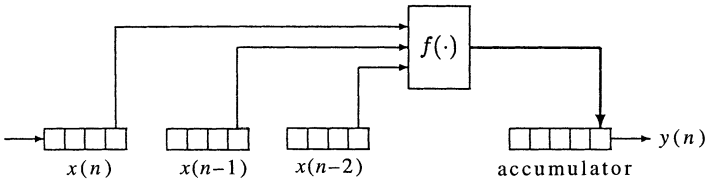

FIGURE 8.4: Distributed arithmetic by Table Lookup. In this example, a sequence $x(n)$ is filtered with a length 3 FIR filter. The wordlength for $x(n)$ is 4 b. The function $f(\cdot)$ is a function of three binary variables, and can be implemented by table lookup. The bits of $x(n)$ are shifted, bit by bit, through the input registers. Accordingly, the bits of $y(n)$ are shifted through the accumulator — after 4 b shifts, a new output $y(n)$ becomes available.

Distributed arithmetic with table lookup can be used with FIR and IIR filters and can be arranged in direct, transpose, cascade, parallel, etc. structures. It can be organized for serial or parallel calculations or for combinations of the two. Because most microprocessors or DSP chips do not have appropriate instructions or architectures for distributed arithmetic, it is best suited for special purpose VLSI design and in those cases, it can be extremely fast.

An alternative realization of these ideas can be developed using a form of periodically time varying system that is oversampled [10].

8.8 Fast Convolution by Number Theoretic Transforms

If one performs all calculations in a finite field or ring of integers rather than the usual infinite field of real or complex numbers, a very efficient type of Fourier transform can be formulated that requires no floating point operations — it supports exact convolution with finite precision arithmetic [1, 2, 17, 26]. This is particularly interesting because a digital computer is a finite machine and arithmetic over finite systems fits it perfectly. In the following, all arithmetic operations are performed modulo some integer M, called the modulus. A bit of number theory can be found in [17, 20, 28].

8.8.1 Number Theoretic Transforms

Here we look at the conditions placed on a general linear transform in order for it to support cyclic convolution. The form of a linear transformation of a length-N sequence of number is given by

$$X(k) = \sum_{n=0}^{N-1} t(n,k)\, x(n) \bmod M \tag{8.51}$$

for $k = 0, 1, \cdots, (N-1)$. The definition of cyclic convolution of two sequences in Z_M is given by

$$y(n) = \sum_{m=0}^{N-1} x(m)\, h(n-m) \bmod M \tag{8.52}$$

for $n = 0, 1, \cdots, (N-1)$ where all indices are evaluated modulo N. We would like to find the properties of the transformation such that it will support cyclic convolution. This means that if $X(k)$, $H(k)$, and $Y(k)$ are the transforms of $x(n)$, $h(n)$, and $y(n)$ respectively, then

$$Y(k) = X(k)\, H(k)\,. \tag{8.53}$$

The conditions are derived by taking the transform defined in (8.1) of both sides of Eq. (8.52) which gives the form for our general linear transform (8.51) as

$$X(k) = \sum_{n=0}^{N-1} \alpha^{nk}\, x(n) \tag{8.54}$$

where α is a *root of order* N, which means that N is the smallest integer such that $\alpha^N = 1$.

THEOREM 8.1 *The transform (8.11) supports cyclic convolution if and only if α is a root of order N and N^{-1} mod M is defined.*

This is discussed in [1, 2]. This transform supports N-point cyclic convolution only if a particular relationship between the modulus M and the data length N is satisfied. The following theorem describes that relationship.

THEOREM 8.2 *The transform (8.11) supports N-point cyclic convolution if and only if*

$$N | O(M) \tag{8.55}$$

where

$$O(M) = gcd\{p_1 - 1,\, p_2 - 1, \cdots,\, p_l - 1\} \tag{8.56}$$

and the prime factorization of M is

$$M = p_1^{r_1}\, p_2^{r_2} \cdots p_l^{r_l}\,. \tag{8.57}$$

Equivalently, N must divide $p_i - 1$ for every prime p_i dividing M. This theorem is a more useful form of Theorem 8.1. Notice that $N_{max} = O(M)$.

One needs to find appropriate N, M, and α such that

- N should be appropriate for a fast algorithm and handle the desired sequence lengths.
- M should allow the desired dynamic range of the signals and should allow simple modular arithmetic.
- α should allow a simple multiplication for $\alpha^{nk}\, x(n)$.

We see that if M is even, it has a factor of 2 and, therefore, $O(M) = N_{max} = 1$ which implies M should be odd. If M is prime the $O(M) = M - 1$ which is as large as could be expected in a field of M integers. For $M = 2^k - 1$, let k be a composite $k = pq$ where p is prime. Then $2^p - 1$ divides $2^{pq} - 1$ and the maximum possible length of the transform will be governed by the length possible for $2^p - 1$. Therefore, only the prime k need be considered interesting. Numbers of this form are know as Mersenne numbers and have been used by Rader [26]. For Mersenne number transforms, it can be shown that transforms of length at least $2p$ exist and the corresponding $\alpha = -2$. Mersenne number transforms are not of as much interest because $2p$ is not highly composite and, therefore, we do not have FFT-type algorithms.

For $M = 2^k + 1$ and k odd, 3 divides $2^k + 1$ and the maximum possible transform length is 2. Thus, we consider only even k. Let $k = s2^t$, where s is an odd integer. Then 2^{2^t} divides $2^{s2^t} + 1$ and the length of the possible transform will be governed by the length possible for $2^{2^t} + 1$. Therefore, integers of the form $M = 2^{2^t} + 1$ are of interest. These numbers are known as Fermat numbers [26]. Fermat numbers are prime for $0 \le t \le 4$ and are composite for all $t \ge 5$.

Since Fermat numbers up to F_4 are prime, $O(F_t) = 2^b$ where $b = 2^t$ and $t \leq 4$, we can have a Fermat number transform for any length $N = 2^m$ where $m \leq b$. For these Fermat primes the integer $\alpha = 3$ is of order $N = 2^b$ allowing the largest possible transform length. The integer $\alpha = 2$ is of order $N = 2b = 2^{t+1}$. Then all multiplications by powers of α are bit shifts — which is particularly attractive because in (8.54), the data values are multiplied by powers of α.

Table 8.2 gives possible parameters for various Fermat number moduli.

TABLE 8.2 Fermat Number Transform Moduli

t	b	$M = F_t$	N_2	$N_{\sqrt{2}}$	N_{max}	α for N_{max}
3	8	2^8+1	16	32	256	3
4	16	$2^{16}+1$	32	64	65536	3
5	32	$2^{32}+1$	64	128	128	$\sqrt{2}$
6	64	$2^{64}+1$	128	256	256	$\sqrt{2}$

This table gives values of N for the two most important values of α which are 2 and $\sqrt{2}$. The second column gives the approximate number of bits in the number representation. The third column gives the Fermat number modulus, the fourth is the maximum convolution length for $\alpha = 2$, the fifth is the maximum length for $\alpha = \sqrt{2}$, the sixth is the maximum length for any α, and the seventh is the α for that maximum length. Remember that the first two rows have a Fermat number modulus which is prime and the second two rows have a composite Fermat number as modulus. Note the differences.

The number theoretic transform itself seems to be very difficult to interpret or use directly. It seems to be useful only as a means for high-speed convolution where it has remarkable characteristics. The books, articles, and presentations that discuss NTT and related topics are [4, 17, 21]. A recent book discusses NT in a signal processing context [14].

8.9 Polynomial-Based Methods

The use of polynomials in representing elements of a digital sequence and in representing the convolution operation has led to the development of a family of algorithms based on the fast polynomial transform [4, 16, 21]. These algorithms are especially useful for two-dimensional convolution. The Chinese remainder theorem for polynomials (CRT), which is central to Winograd's short convolution algorithm, is also conveniently described in polynomial notation. An interesting approach combines the use of the polynomial-based methods with the number theoretic approach to convolution (NTTs), wherein the elements of a sequence are taken to lie in a finite field [9, 15]. In [15] the CRT is extended to the case of a ring of polynomials with coefficients from a finite ring of integers. It removes the limitations on both word length and sequence length of NNTs and serves as a link between the two methods (CRT and NNT). The new result so obtained, which specializes to both the NNTs and the CRT for polynomials, has been called the AICE-CRT (the American-Indian-Chinese extension of the CRT). A complex version has also been derived.

8.10 Special Low-Multiply Filter Structures

In the use of convolution for digital filtering, the convolution operation can be simplified, if the filter $h(n)$ is chosen appropriately.

Some filter structures are especially simple to implement. Some examples are:

- A simple implementation of the recursive running sum (RRS) is based on the factorization

$$\sum_{k=0}^{L-1} z^k = (z^L + 1)/(z - 1).$$

- If the transfer function $H(z)$ of the filter possesses a root at $z = -1$ of multiplicity K, the factor $(z + 1)/2$ can be extracted from the transfer function. The factor $(z + 1)/2$ can be implemented very simply.
- This idea is extended in prefiltering and IFIR filtering techniques — a filter is implemented as a cascade of two filters: one with a crude response that is simple to implement, another that makes up for it, but requires the usual implementation complexity. The overall response satisfies specifications and can be implemented with reduced complexity.
- The maximally flat symmetric FIR filter can be implemented without multiplications using the De Casteljau algorithm [27].

In summary, a filter can often be designed so that the convolution operation can be performed with less computational complexity and/or at a faster rate. Much work has focused on methods that take into account implementation complexity during the approximation phase of the filter design process. (See the chapter on digital filter design).

References

[1] Agarwal, R.C. and Burrus, C.S., Fast convolution using Fermat number transforms with applications to digital filtering, *IEEE Trans. Acoustics Speech Signal Process.*, ASSP-22(2):87–97, April, 1974. Reprinted in [17].

[2] Agarwal, R.C. and Burrus, C.S., Number theoretic transforms to implement fast digital convolution, *Proc. IEEE*, 63(4):550–560, April, 1975. (Also in IEEE Press DSP Reprints II, 1979).

[3] Agarwal, R.C. and Cooley, J.W., New algorithms for digital convolution, *IEEE Trans. Acoustics Speech Signal Process.*, 25(5):392–410, October, 1977.

[4] Blahut, R.E. *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, MA, 1985.

[5] Burrus, C.S., Block implementation of digital filters, *IEEE Trans. Circuit Theory*, CT-18(6):697–701, November, 1971.

[6] Burrus, C.S., Block realization of digital filters, *IEEE Trans. Audio Electroacoust.*, AU-20(4):230–235, October, 1972.

[7] Burrus, C.S., Digital filter structures described by distributed arithmetic, *IEEE Trans. Circuits Syst.*, CAS-24(12):674–680, December, 1977.

[8] Burrus, C.S., Efficient Fourier transform and convolution algorithms, in Jae S. Lim and Alan V. Oppenheim, Eds., *Advanced Topics in Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] Garg, H.K., Ko, C.C., Lin, K.Y., and Liu, H., On algorithms for digital signal processing of sequences, *Circuits Syst. Signal Process.*, 15(4):437–452, 1996.

[10] Ghanekar, S.P., Tantaratana, S., and Franks, L.E., A class of high-precision multiplier-free FIR filter realizations with periodically time-varying coefficients, *IEEE Trans. Signal Process.*, 43(4):822–830, 1995.

[11] Gold, B. and Rader, C.M., *Digital Processing of Signals*, McGraw-Hill, New York, 1969.

[12] Harris, F.J., Time domain signal processing with the DFT, in D. F. Elliot, ed., *Handbook of Digital Signal Processing*, ch. 8, 633–699, Academic Press, NY, 1987.

[13] Helms, H.D., Fast Fourier transform method of computing difference equations and simulating filters, *IEEE Trans. Audio Electroacoust.*, AU-15:85–90, June, 1967.

[14] Krishna, H., Krishna, B., Lin, K.-Y, and Sun, J.-D., *Computational Number Theory and Digital Signal Processing*, CRC Press, Boca Raton, FL, 1994.

[15] Lin, K.Y., Krishna, H., and Krishna, B., Rings, fields the Chinese remainder theorem and an American-Indian-Chinese extension, part I: Theory. *IEEE Trans. Circuits Syst. II*, 41(10):641–655, 1994.

[16] Loh, A.M. and Siu, W.-C., Improved fast polynomial transform algorithm for cyclic convolutions, *Circuits Syst. Signal Process.*, 14(5):603–614, 1995.

[17] McClellan, J.H. and Rader, C.M., *Number Theory in Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[18] Mou, Z.-J. and Duhamel, P., Short-length FIR filters and their use in fast nonrecursive filtering, *IEEE Trans. Signal Process.*, 39(6):1322–1332, June, 1991.

[19] Myers, D.G., *Digital Signal Processing: Efficient Convolution and Fourier Transform Techniques*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[20] Niven, I. and Zuckerman, H.S., *An Introduction to the Theory of Numbers*, 4th ed., John Wiley & Sons, New York, 1980.

[21] Nussbaumer, H.J., *Fast Fourier Transform and Convolution Algorithms*, Springer-Verlag, New York, 1982.

[22] Oppenheim, A.V. and Schafer, R.W., *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[23] Phoong, S- M. and Vaidyanathan, P.P., One- and two-level filter-bank convolvers, *IEEE Trans. Signal Process.*, 43(1):116–133, January, 1995.

[24] Proakis, J.G., Rader, C.M., Ling, F., and Nikias, C.L., *Advanced Digital Signal Processing*, Macmillan, New York, 1992.

[25] Rabiner, L.R. and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[26] Rader, C.M., Discrete convolution via Mersenne transforms, *IEEE Trans. Comput.*, 21(12):1269–1273, December, 1972.

[27] Samadi, S., Cooklev, T., Nishihara, A., and Fujii, N., Multiplierless structure for maximally flat linear phase FIR filters, *Electron. Lett.*, 29(2):184–185, Jan. 21, 1993.

[28] Schroeder, M.R., *Number Theory in Science and Communication*, 2nd ed., Springer-Verlag, Berlin, 1984, 1986.

[29] Selesnick, I.W. and Burrus, C.S., Automatic generation of prime length FFT programs, *IEEE Trans. Signal Process.*, 44(1):14–24, January, 1996.

[30] Stockham, T.G., High speed convolution and correlation, in *AFIPS Conf. Proc.*, vol. 28, pp. 229–233, Spring Joint Computer Conference, 1966.

[31] Tolimieri, R., An, M., and Lu, C., *Algorithms for Discrete Fourier Transform and Convolution*, Springer-Verlag, New York, 1989.

[32] Vaidyanathan, P.P, Orthonormal and biorthonormal filter banks as convolvers, and convolutional coding gain, *IEEE Trans. Signal Process.*, 41(6):2110–2129, June, 1993.

[33] Vetterli, M., Running FIR and IIR filtering using multirate filter banks, *IEEE Trans. Acoust. Speech Signal Process.*, 36(5):730–738, May, 1988.

[34] White, S.A., Applications of distributed arithmetic to digital signal processing, *IEEE ASSP Mag.*, 6(3):4–19, July, 1989.

[35] Winograd, S., *Arithmetic Complexity of Computations*, *SIAM*, 1980.

[36] Zalcstein, Y., A note on fast cyclic convolution, *IEEE Trans. Comput.*, 20:665–666, June, 1971.