Chapter 3: Survey of Convolution Algorithms and Techniques

This chapter surveys algorithms for linear and cyclic convolution in a form that is convenient for automatic generation. All of the algorithms are presented using the uniform mathematical notation of bilinear algorithms and are derived systematically using polynomial algebra and properties of the tensor product. Algorithms implicitly refer to bilinear algorithms, and operations on bilinear algorithms use the definitions in Section 2.3.

3.1 Linear Convolution

3.1.1 Standard Algorithm

In a few rare cases, the standard method of multiplying polynomials learned in high school might be the best choice for a linear convolution algorithm. This can be turned into a bilinear algorithm of matrices in the obvious way.

Example 4 *A* 3×3 *linear convolution given by the Standard Algorithm is :*

$$\begin{aligned} sb_3 &= \left(\begin{bmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0&0 \\ 0&0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{bmatrix}, \begin{bmatrix} 1&0&0 \\ 0&1&0 \\ 1&0&0 \\ 0&0&1 \\ 0&1&0 \\ 1&0&0 \\ 0&0&1 \\ 0&1&0 \\ 0&0&1 \end{bmatrix}, \begin{bmatrix} 1&0&0 \\ 1&0&0 \\ 0&1&0 \\ 1&0&0 \\ 0&1&0 \\ 0&0&1 \\ 0&1&0 \\ 0&0&1 \\ 0&0&1 \end{bmatrix} \right) \\ &= (sb_3[C], sb_3[A], sb_3[B]) \end{aligned}$$

3.1.2 Toom-Cook Algorithm

The Toom-Cook algorithm [28, 7, 15] uses evaluation and interpolation to compute the product of two polynomials. To compute the product $h(x) = f(x)g(x)$, where f and g are $N-1$ degree polynomials, first evaluate each polynomial at $2N-1$ distinct values α_i. Next compute the $2N-1$ multiplications $h(\alpha_i) = f(\alpha_i)g(\alpha_i)$. Finally, use the $2N-1$ points $(\alpha_i, h(\alpha_i))$ and the Lagrange interpolation formula to recover

$$h(x) = \sum_{j=0}^{2N-2} h(\alpha_i) \prod_{k \neq j} \frac{x - \alpha_k}{\alpha_j - \alpha_k}.$$

This algorithm can be expressed as a bilinear algorithm using the following notation.

Definition 5 (Bar Notation)
Let $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ *we will denote by* $\overline{A(x)}$ *the equivalent vector* $\begin{bmatrix} a_0 & a_1 & \ldots & a_n \end{bmatrix}^T$

Definition 6 (Vandermonde Matrix)

$$\mathbf{V}[\alpha_0, \ldots, \alpha_n] = \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \ldots & \alpha_0^n \\ 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^n \end{bmatrix}.$$

The matrix $\mathbf{V}$ applied to the vector of coefficients of $f(x)$ is equal to the vector containing the evaluations $f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_n)$, and applying $\mathbf{V}^{-1}$ to the vector of evaluations returns the original coefficients. Therefore $\mathbf{V}^{-1}$ corresponds to interpolation and can be computed using Lagrange's formula. The following theorem summarizes these observations.

Theorem 7 (Toom-Cook Algorithm) *The bilinear algorithm* $(\mathbf{V}^{-1}, \mathbf{V}', \mathbf{V}')$*, where* $\mathbf{V}'$ *is the* $(2N-1) \times N$ *matrix containing the first* N *columns of* $V[\alpha_0, \ldots, \alpha_{2N-1}]$*, computes the* N*-point linear convolution of two vectors.*

This theorem is a special case of Theorem 3 and follows from the Chinese Remainder theorem applied to $f(x) = \prod_{i=0}^{2N-1}(x - \alpha_i)$. The matrix R in this case is the Vandermonde matrix $V[\alpha_0, \ldots, \alpha_{2N-1}]$.

The Toom-Cook algorithm reduces the number of "general" multiplications from N^2 (computed by definition) to $2N-1$ at the cost of more additions. A general multiplication is one that cannot be precomputed at compile time, or reduced to a series of additions at run-time. For small input sizes when there are sufficiently many convenient evaluation points such as $0, 1, -1, \infty$, then the reduction in general multiplications corresponds to a reduction in actual multiplications. What is meant by evaluating at ∞ is if $f(x) = f_0 + f_1x + \ldots + f_kx^k$, with f_k non-zero, then $f(\infty) = f_k$. (To see why this makes sense, consider the limit of $f(x)/f_kx^k$ as x tends to infinity.)

Example 3 corresponds to the Toom-Cook algorithm using evaluation points 0, 1, and ∞; the following 3 point example uses evaluation points 0, 1, -1, 2, and ∞.

Example 5 *A* 3×3 *linear convolution given by the Toom-Cook algorithm is:*

$$
\begin{aligned}
tc_3 &= \left(\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1/2 & 1 & -1/3 & -1/6 & 2 \\ -1 & 1/2 & 1/2 & 0 & -1 \\ 1/2 & -1/2 & -1/6 & 1/6 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \right) \\
&= (tc_3[C], tc_3[A], tc_3[B])
\end{aligned}
$$

Note further that the algorithm can be improved to use fewer operations by using:

$$
\begin{aligned}
tc_3[A] = tc_3[B] &= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

3.1.3 Combining Linear Convolutions

The tensor product can be used to combine small linear convolution algorithms into larger ones in an efficient manner. This is important, because the tensor product of smaller convolution algorithms will generally use fewer operations than a direct larger convolution algorithm. For example combining a Toom-Cook algorithm of size 2 with a Toom-Cook algorithm of size 3, creates a linear convolution of size 6 that uses many fewer (62 versus 114 for real inputs with one vector fixed) operations than a Toom-Cook convolution of size 6.

Theorem 8 (Tensor Product of Linear Convolutions) *Let $\mathcal{L}_m$ and $\mathcal{L}_n$ be bilinear algorithms for linear convolution of size m and n respectively. Then $O_{m,n}(\mathcal{L}_m \otimes \mathcal{L}_n)$ is a bilinear algorithm for linear convolution of size mn, where $O_{m,n}$ is a sparse $(2m-1)(2n-1) \times (2mn-1)$ matrix. The non-zero entries are equal to one and occur in locations $jm+i, j(2m-1)+i$ and $jm+i, (j-1)(2m-1)+m+i$ for $0 \leq j < 2n-1$ and $0 \leq i < m-1$.*

The proof is most easily seen from the polynomial interpretation of convolution. Let $a(x)$ and $b(x)$ be polynomials of degree $mn-1$, and let

$$A(x,y) = \sum_{i=0}^{n-1} A_i(x) y^i \text{ and } B(x,y) = \sum_{j=0}^{n-1} B_j(x) y^j,$$

where $A_i(x)$ and $B_j(x)$ are polynomials of degree $m-1$. Next, substitute $y = x^m$, $a(x) = A(x, x^m)$ and $b(x) = B(x, x^m)$. Consequently, if $C(x, y) = A(x, y)B(x, y)$, then $c(x) = C(x, x^m)$. By Lemma 1 and Example 2, $\mathcal{L}_m \otimes \mathcal{L}_n$ computes $C(x, y)$. The matrix $O_{m,n}$ corresponds to the reduction obtained from substituting $y = x^m$ into $C(x, y)$.

Example 6

$$O_{2,3} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

The following generalization is obtained using induction and simple properties of the tensor product.

Theorem 9 *Let $N = n_1, \ldots, n_t$ and let $\mathcal{L}_{n_i}$, $0 \le i < t$ be linear convolution algorithms of size n_i. Then $O_{n_1,\ldots,n_t}(\mathcal{L}_{n_1} \otimes \cdots \otimes \mathcal{L}_{n_t}) = \mathcal{L}_{n_1 \cdots n_t}$, where $O_{n_1,\ldots,n_t}$ is a sparse $(2n_1-1)\cdots(2n_t-1) \times (2N-1)$ matrix defined by $O_{n_1,\ldots,n_t} = O_{n_1,n_2\cdots n_t}(I_{2n_1-1} \otimes O_{n_2,\ldots,n_t})$.*

3.2 Linear Convolution via Cyclic Convolution

Tolimieri in [27] points out that linear convolution can be obtained from generalized cyclic convolution corresponding to polynomial multiplication modulo a polynomial. For example, if $g(x) = g_0 + g_1 x + g_2 x^2$ and $h(x) = h_0 + h_1 x + h_2 x^2$, then $g(x)h(x)$ can be computed by first convolving g and h via a 4-point cyclic convolution and then adding the vector $g_2 h_2 \overline{m(x)}$ where $m(x) = x^4 - 1$. The following theorem expresses Tolimieri's method in terms of bilinear algorithms.

Theorem 10 (Linear from Cyclic) *Let $g(x)$, $h(x)$ be polynomials of degree $n-1$ and $m(x) = x^{2n-2} + \sum_{i=0}^{2n-3} m_i x^i$, be a monic polynomial of degree $2n-2$. Assume that (C_m, A_m, B_m) is a bilinear algorithm that computes $\overline{g(x)h(x) \mod m(x)}$. Then the bilinear algorithm (C, A, B) computes $f(x)g(x)$, where*

$$\begin{aligned}
C &= \begin{bmatrix} 1 & & & m_0 \\ & \ddots & & \vdots \\ & & 1 & m_{2n-3} \\ & & & 1 \end{bmatrix} \begin{bmatrix} C_m & \\ & 1 \end{bmatrix}, \\
A &= \begin{bmatrix} A_m & \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ & & 1 \end{bmatrix}, \\
B &= \begin{bmatrix} B_m & \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ & & 1 \end{bmatrix}
\end{aligned}$$

Proof

Let $c(x) = \overline{g(x)h(x) \mod m(x)}$. Therefore, $f(x)g(x) = c(x) + q(x)m(x)$, and since $m(x)$ is monic and of degree $2n-2$, $g(x)h(x) = c(x) + g_n h_n m(x)$.

$$\begin{aligned}
(C, A, B)\,(g, h) &= \begin{bmatrix} 1 & & & m_0 \\ & \ddots & & \vdots \\ & & 1 & m_{2n-3} \\ & & & 1 \end{bmatrix} \begin{bmatrix} C_m\,(A_m g \bullet B_m h) \\ g_n h_n \end{bmatrix} \\
&= \overline{g(x)h(x) \mod m(x)} + \overline{g_n h_n m(x)} \\
&= \overline{g(x)h(x)}.
\end{aligned}$$

3.3 Cyclic Convolution

Convolution modulo $f(x)$ refers to polynomial multiplication modulo a third polynomial. Algorithms for convolution modulo $f(x)$ can be obtained from linear convolution algorithms by multiplying by a matrix, which corresponds to computing the remainder in division by $f(x)$. Let $M(f(x))$ denote the reduction matrix defined by $M(f(x))\overline{A(x)} = \overline{A(x) \mod f(x)}$. The exact form of $M(f(x))$ depends on the degree of $A(x)$. If (C, A, B) is a bilinear algorithm for linear convolution, then $(M(f(x))C, A, B)$ is a bilinear algorithm for convolution modulo $f(x)$.

Example 7 *Composing*

$$M(x^2 - 1) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

with the Toom-Cook bilinear algorithm of equation 2.6, the bilinear algorithm

$$(M(x^2-1)C_2, A_2, B_2) = \left(\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \right)$$

for 2-point cyclic convolution is obtained.

3.3.1 Convolution Theorem

The well-known convolution theorem provides a bilinear algorithm for computing cyclic convolution.

Theorem 11 (Convolution Theorem)
The bilinear algorithm $(\mathrm{DFT}_N^{-1}, \mathrm{DFT}_N, DFT_N)$ *computes* N*-point cyclic convolution.*

Proof
Let ω_N be a primitive N-th root of unity, then $x^N - 1 = \prod_{i=0}^{N-1}(x - \omega_N^i)$. Since, $\mathbf{V}[1, \omega_N, \ldots, \omega_N^{N-1}] = \mathrm{DFT}_N$, the convolution theorem follows from Theorem 3.

When $N = RS$, $x^N - 1 = \prod_{i=0}^{S-1}(x^R - \omega_S^i)$. Applying the Chinese Remainder theorem to this factorization leads to the following theorem which allows the DFT to be combined with other convolution algorithms.

Theorem 12 *Let* $N = RS$ *and let* $\mathcal{C}_i$, $i = 0, \ldots, S-1$, *be bilinear algorithms to multiply two polynomials modulo* $x^R - \omega_S^i$. *Then*

$$(\mathrm{DFT}_S^{-1} \otimes I_R) \left(\bigoplus_{i=0}^{S-1} \mathcal{C}_i \right) (\mathrm{DFT}_S \otimes I_R)$$

is a bilinear algorithm to compute N*-point convolution.*

Proof
Let $f(x)$ be a polynomial of degree $N-1$ and write $f(x) = \sum_{j=0}^{S-1} f_j(x) x^{Rj}$, where $\deg(f_j(x)) < R$. Then $f(x) \mod x^R - \omega_S^i = \sum_{j=0}^{S-1} f_j(x)\omega_S^j$. Therefore, the matrix $R = [R_0\ R_1\ \ldots R_{S-1}]^T$ with $R_i f = f(x) \mod x^R - \omega_j^i$ is equal to $\mathrm{DFT}_S \otimes I_R$.

Note that multiplication modulo $x^R - \alpha$ can easily be transformed into cyclic convolution. Observe that if $\beta^R = \alpha$, and $h(x) = f(x)g(x) \mod x^R - \alpha$, then

$$\begin{aligned} h_\beta(x) &= h(\beta x) = f(\beta x)g(\beta x) \pmod{(\beta x)^R - \alpha} \\ &= f(\beta x)g(\beta x) \pmod{(\beta x)^R - \alpha} \\ &= f(\beta x)g(\beta x) \pmod{\alpha(x^R - 1)}. \end{aligned}$$

Therefore, $h(x) = h_\beta(x/\beta)$.

Applying this observation and the previous theorem leads to the following construction related to the FFT shown in (2.7).

Theorem 13 *Let $\mathcal{C}_R$ be a bilinear algorithm to compute R-point cyclic convolution, and let $\mathcal{F}_S = ((\mathrm{DFT}_S \otimes I_R), T_R^N(\mathrm{DFT}_S \otimes I_R), T_R^N(\mathrm{DFT}_S \otimes I_R))$. Then $(I_S \otimes \mathcal{C}_R)\mathcal{F}_S$ computes N-point cyclic convolution.*

The following example uses Theorem 12 and the matrix exchange theorem to obtain a result that is similar to Theorem 13 but without the need for Twiddle factors.

Example 8 *Let F_n represent a size n FFT, and let $\alpha = e^{2\pi i/n}$. Now let $\mathcal{L}_m = (C_m, A_m, B_m)$ represent a linear convolution of size m. From Theorem 12 a size mn cyclic convolution is computed by*

$$(F_n^{-1} \otimes I_m)\left(\bigoplus_{i=0}^{n-1} M(x^m - \alpha^i)\mathcal{L}_m\right)(F_n \otimes I_m) \tag{3.1}$$

Now suppose (C, A, B) is the bilinear algorithm representing the size mn cyclic convolution of (3.1), then

$$\begin{aligned}
C &= \; = (F_n^{-1} \otimes I_m)\left(\bigoplus_{i=0}^{n-1} M(x^m - \alpha^i)C_m\right) \\
A &= \left(\bigoplus_{i=0}^{n-1} A_m\right)(F_n \otimes I_m) \\
&= (I_n \otimes A_m)(F_n \otimes I_m) \\
B &= \left(\bigoplus_{i=0}^{n-1} B_m\right)(F_n \otimes I_m) \\
&= (I_n \otimes B_m)(F_n \otimes I_m)
\end{aligned}$$

Note that the direct sums for A and B can be changed to tensor products because A_m and B_m do not change with i, (this of course is not true for C).

After applying matrix exchange the following theorem has just been derived and proved.

Theorem 14 (Mixed Convolution Theorem)

Let $\alpha = e^{2\pi i/n}$, F_n be a size n FFT, and (C_m, A_m, B_m) be a size m linear convolution. Then

$$J \cdot (F_n \otimes I_m)(I_n \otimes A_m^T) \cdot D \cdot (I_n \otimes B_m)(F_n \otimes I_m)$$

is a size mn cyclic convolution, where

$$D = \left(\bigoplus_{i=0}^{n-1} M(x^m - \alpha^i) C_m \right)^T \cdot (F_n^{-1} \otimes I_m) \cdot J\mathbf{v},$$

J is the anti-identity matrix, and $\mathbf{v}$ is a fixed input vector.

3.3.2 Winograd Convolution Algorithm

Winograd's algorithm [29] for computing cyclic convolution follows from the Chinese Remainder Theorem when applied to the irreducible rational factors of the polynomial $X^N - 1$. The irreducible rational factors of $x^N - 1$ are called cyclotomic polynomials.

Definition 7 (Cyclotomic Polynomials)

The cyclotomic polynomials can be defined recursively from the formula

$$x^N - 1 = \prod_{d|N} \Phi_d(x).$$

Alternatively

$$\Phi_N(x) = \prod_{\gcd(j,N)=1} (x - \omega_N^j),$$

where ω_N is a primitive N-th root of unity. It follows that $\deg(\Phi_N(x)) = \phi(N)$, where ϕ is the Euler ϕ function. It is well known [16] that $\Phi_N(x)$ has integer coefficients and is irreducible over the rationals.

Applying Theorem 3 to $x^N - 1 = \prod_{d|N} \Phi_d(x)$ leads to the following algorithm.

Theorem 15 (Winograd Convolution Algorithm)

Let $\mathcal{C}_f$ denote a bilinear algorithm that multiplies elements of $\mathbb{C}[x]/f(x)$. Then

$$R^{-1} \left(\bigoplus_{d|n} \mathcal{C}_{\Phi_d(x)} \right) R \tag{3.2}$$

where $R = [R_{d_1}\ R_{d_2}\ \ldots R_{d_k}]^T$ and $R_{d_i}\overline{f} = \overline{f(x) \bmod \Phi_{d_i}(x)}$ is a bilinear algorithm for N-point cyclic convolution.

Using the 2-point cyclic convolution algorithm in Example 7 and the cyclotomic polynomials $\Phi_1(x) = (x-1)$, $\Phi_2(x) = (x+1)$, and $\Phi_4(x) = (x^2+1)$ the following 4-point cyclic convolution algorithm is obtained.

Example 9

$$\left(R_4^{-1} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & 0 & -1 \\ & & -1 & 1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & 0 \\ & & 1 & 1 \\ & & 0 & 1 \end{bmatrix} R_4, \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & 0 \\ & & 1 & 1 \\ & & 0 & 1 \end{bmatrix} R_4 \right),$$

where R_4 is the R matrix in Example 1.

The results in the next section provide a more efficient method for computing R_4.

3.3.3 CRT-Based Cyclic Convolution Algorithms for Prime Powers

Selesnick and Burrus [22] have shown that when $N = p^k$ is a prime power, the Winograd algorithm has additional structure. This structure follows from the properties

$$\begin{aligned} \Phi_p(x) &= x^{p-1} + \cdots + x + 1 \\ \Phi_{p^k}(x) &= \Phi_p(x^{p^{k-1}}). \end{aligned}$$

The composition structure of $\Phi_{p^k}(x)$ provides an efficient way to compute R_{p^k}.

Theorem 16 *Let $R_{p^k} = [R_0, R_p, \ldots, R_{p^k}]^t$ be the $p^k \times p^k$ reduction matrix where $R_{p^i}\overline{f(x)} = \overline{f(x) \bmod \Phi_{p^i}(x)}$ for $f(x)$ of degree $p^k - 1$. Then*

$$R_{p^k} = \begin{bmatrix} 1_p \otimes R_{p^{k-1}} \\ G_p \otimes I_{p^{k-1}} \end{bmatrix},$$

where G_n is the $(n-1) \times n$ matrix:

$$G_n = \begin{bmatrix} 1 & & & -1 \\ & 1 & & -1 \\ & & \ddots & -1 \\ & & 1 & -1 \end{bmatrix},$$

and 1_n is the $1 \times n$ matrix filled with $1's$. Moreover, $R_{p^k} = (R_{p^{k-1}} \oplus I_{(p-1)p^{k-1}})(R_p \otimes I_{p^{k-1}})$.

Proof

First observe that if $f(x) = f_0 + f_1 x + \cdots f_{m-1}x^{m-1} + x^m$ and $A(x) = \sum_{i=0}^{m} a_i x^i$, then $A(x)$ mod $f(x) = \sum_{i=0}^{m-1}(a_i - f_i)x^i$. Therefore reduction of $A(x)$ modulo $f(x)$ is given by

$$R = \begin{bmatrix} 1 & & & -f_0 \\ & \ddots & & -f_1 \\ & & 1 & -f_{m-1} \end{bmatrix}.$$

When $f(x) = 1 + x + \cdots + x^{n-1}$ the matrix G_n is obtained. Next, observe that if $A(x) = \sum_{i=0}^{m} A_i(x)x^{ni}$, where $\deg(A_i) < n$, then $A(x) \mod f(x^n) = \sum_{i=0}^{m}(A_i(x) - f_i A_m(x))x^{ni}$. Therefore reduction of $A(x) \mod f(x^n)$ is given by $R \otimes I_n$, and reduction modulo $\Phi_{p^k}(x) = \Phi_p(x^{p^{k-1}})$ is given by $G_p \otimes I_{p^{k-1}}$. Finally, since $x^{p^{k-1}} \mod \Phi_{p^{k-1}} = 1$, reduction of $A(x)$ modulo $\{\Phi_{p^i}(x), i = 0, \ldots, k\}$ is given by $1_p \otimes R_{p^{k-1}}$. These observations prove the first part of the theorem. The factorization in the second part is obtained using the multiplicative property of the tensor product.

A simple block matrix multiplication provides the following computation of the inverse of R_{p^k}.

Theorem 17

$$R_{p^k}^{-1} = 1/p \begin{pmatrix} 1_p^t \otimes R_{p^{k-1}}^{-1} & V_p \otimes I_{p^{k-1}} \end{pmatrix},$$

where V_n is the $n \times (n-1)$ matrix

$$\begin{bmatrix} n-1 & -1 & -1 & \ldots & -1 \\ -1 & n-1 & -1 & \ldots & -1 \\ \vdots & & \ddots & & \vdots \\ -1 & \ldots & -1 & n-1 & -1 \\ -1 & \ldots & -1 & -1 & n-1 \\ -1 & \ldots & -1 & -1 & -1 \end{bmatrix}$$

Moreover, $R_{p^k}^{-1} = (R_p^{-1} \otimes I_{p^{k-1}})(R_{p^{k-1}}^{-1} \oplus I_{(p-1)p^{k-1}})$.

Example 10 *A bilinear algorithm for a cyclic convolution of size 27 is (C, A, B), where $\mathcal{L}_n = (\mathcal{L}_n[C], \mathcal{L}_n[A], \mathcal{L}_n[B])$ is a bilinear algorithm for a linear convolution of size n of any method, and*

$$\begin{aligned} C &= R_{3^3}^{-1} \begin{bmatrix} 1 & & & \\ & M(x^2+x+1)\mathcal{L}_2[C] & & \\ & & M(x^6+x^3+1)\mathcal{L}_6[C] & \\ & & & M(x^{18}+x^9+1)\mathcal{L}_{18}[C] \end{bmatrix}, \\ A &= \begin{bmatrix} 1 & & & \\ & \mathcal{L}_2[A] & & \\ & & \mathcal{L}_6[A] & \\ & & & \mathcal{L}_{18}[A] \end{bmatrix} R_{3^3}, \\ B &= \begin{bmatrix} 1 & & & \\ & \mathcal{L}_2[B] & & \\ & & \mathcal{L}_6[B] & \\ & & & \mathcal{L}_{18}[B] \end{bmatrix} R_{3^3}, \end{aligned}$$

and

$$R_{3^3} = \begin{bmatrix} \begin{matrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{matrix} & \\ & I_{24} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \otimes I_3 & \\ & I_{18} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \otimes I_9 \end{bmatrix}$$

3.3.4 The Agarwal-Cooley and Split-Nesting Algorithms

The Agarwal-Cooley [1] algorithm uses the tensor product to create a larger cyclic convolution from smaller cyclic convolutions. The Split-Nesting algorithm, due to Nussbaumer [19], follows directly from Agarwal-Cooley using simple properties of the tensor product.

The Agarwal-Cooley algorithm follows from the fact that when $\gcd(m, n) = 1$, the algebra $\mathbb{F}[x]/(x^{mn} - 1)$ is isomorphic to $\mathbb{F}[y, z]/(y^m - 1, z^n - 1)$, which by Example 2 is isomorphic to $\mathbb{F}[y]/(y^m - 1) \otimes \mathbb{F}[z]/(z^n - 1)$. The isomorphism is obtained by mapping x to yz which maps x^i to $y^{(i \mod m)}z^{(i \mod n)}$. Using the reordering required by this mapping and Lemma 1 leads to the following theorem which shows how to build an mn-point cyclic convolution algorithm from the tensor product of m-point and n-point cyclic convolution algorithms.

Theorem 18 (Agarwal-Cooley Algorithm)
Assume $\gcd(m, n) = 1$ *and let* $\mathcal{C}_m = (C_m, A_m, B_m)$ *and* $\mathcal{C}_n = (C_n, A_n, B_n)$ *be bilinear algorithms for cyclic convolution of size* m *and* n. *Let* $Q^{-1}_{m,n}$ *be the permutation that maps* i *to* $(i \mod m)n + (i \mod n)$. *Then* $Q^{-1}_{m,n}(\mathcal{C}_m \otimes \mathcal{C}_n)Q_{m,n}$ *computes a cyclic convolution of size* mn.

The permutation $Q_{m,n}$ is defined by the mapping $in + j \mapsto ie_m + je_n \mod mn$, $0 \leq i < m$, $0 \leq j < n$, where $e_m \equiv 1 \mod m$, $e_m \equiv 0 \mod n$, $e_n \equiv 0 \mod m$, $e_n \equiv 1 \mod n$, are the idempotents defining the Chinese remainder theorem mapping for the integers m and n.

Let $R_m^{-1}\left(\bigoplus_{i=0}^{k_1} \mathcal{C}_{m_i}\right) R_m$ and $R_n^{-1}\left(\bigoplus_{i=0}^{k_2} \mathcal{C}_{n_i}\right) R_n$ be bilinear algorithms to compute m, and n-point Winograd cyclic convolutions. Then combining Agarwal-Cooley with the Winograd algorithm yields the bilinear algorithm

$$Q^{-1}_{m,n}\left(R_m^{-1}\left(\bigoplus_{i=0}^{k_1} \mathcal{C}_{m_i}\right) R_m\right) \otimes \left(R_n^{-1}\left(\bigoplus_{j=0}^{k_2} \mathcal{C}_{n_j}\right) R_n\right) Q_{m,n} \tag{3.3}$$

for computing an mn-point cyclic convolution, (provided $gcd(m, n) = 1$). Using the multiplicative property of the tensor product, this is equal to

$$Q^{-1}_{m,n}(R_m^{-1} \otimes R_n^{-1})\left(\left(\bigoplus_{i=0}^{k_1} \mathcal{C}_{m_i}\right) \otimes \left(\bigoplus_{j=0}^{k_2} \mathcal{C}_{n_j}\right)\right)(R_m \otimes R_n)Q_{m,n}. \tag{3.4}$$

Rearranging this equation into a double sum of tensor products leads to the "Split-Nesting Algorithm" which was first derived by Nussbaumer [19], who observed that it requires fewer additions then equation 3.3. The following theorem describes this transformation.

Theorem 19 (Split Nesting) *Let $\mathcal{C} = \bigoplus_{i=0}^{s-1} \mathcal{C}_i$ and $\mathcal{D} = \bigoplus_{j=0}^{t-1} \mathcal{D}_j$. Then*

$$\mathcal{C} \otimes \mathcal{D} = P^{-1} \left(\bigoplus_{i=0}^{s-1} \bigoplus_{j=0}^{t-1} \mathcal{C}_i \otimes \mathcal{D}_j \right) P,$$

where P is a permutation.

Proof

Using the first part of Theorem 6,

$$\mathcal{C} \otimes \mathcal{D} = \bigoplus_{i=0}^{s-1} \mathcal{C}_i \otimes \bigoplus_{j=0}^{t-1} \mathcal{D}_j = \bigoplus_{i=0}^{s-1} \left(\mathcal{C}_i \otimes \bigoplus_{j=0}^{t-1} \mathcal{D}_j \right).$$

Using the second part of Theorem 6, the previous equation is equal to

$$\bigoplus_{i=0}^{s-1} P_i^{-1} \left(\bigoplus_{j=0}^{t-1} \mathcal{C}_i \otimes \mathcal{D}_j \right) P_i, \text{ which is equal to } P^{-1} \left(\bigoplus_{i=0}^{s-1} \bigoplus_{j=0}^{t-1} \mathcal{C}_i \otimes \mathcal{D}_j \right) P,$$

where $P = \bigoplus_{i=0}^{s-1} P_i$.

Example 11 *Let $\mathcal{C}_4 = R_4^{-1}(1 \oplus 1 \oplus \mathcal{C}_2)R_4$ and $\mathcal{C}_{27} = R_{27}^{-1}(1 \oplus \mathcal{D}_2 \oplus \mathcal{D}_6 \oplus \mathcal{D}_{18})R_{27}$, where $\mathcal{C}_2 = M(x^2+1)\mathcal{L}_2$, $\mathcal{D}_2 = M(x^2+x+1)\mathcal{L}_2$, $\mathcal{D}_6 = M(x^6+x^3+1)\mathcal{L}_6$, $\mathcal{D}_{18} = M(x^{18}+x^9+1)\mathcal{L}_{18}$, are the algorithms for cyclic convolution on 4 and 27 points given in Examples 9 and 10. By Agarwal-Cooley,*

$$Q_{4,27}^{-1}(R_4^{-1}(1 \oplus 1 \oplus \mathcal{C}_2)R_4) \otimes (R_{27}^{-1}(1 \oplus \mathcal{D}_2 \oplus \mathcal{D}_6 \oplus \mathcal{D}_{18})R_{27})Q_{4,27}$$

is an algorithm for cyclic convolution on 108 points. The split nesting theorem transforms this algorithm into

$$\begin{aligned}
&(Q_{4,27}^{-1}(R_4^{-1} \otimes R_{27}^{-1})P^{-1} \\
&(1 \oplus \mathcal{D}_2 \oplus \mathcal{D}_6 \oplus \mathcal{D}_{18}) \oplus (1 \oplus \mathcal{D}_2 \oplus \mathcal{D}_6 \oplus \mathcal{D}_{18}) \oplus (\mathcal{C}_2 \oplus \mathcal{C}_2 \otimes \mathcal{D}_2 \oplus \mathcal{C}_2 \otimes \mathcal{D}_6 \oplus \mathcal{C}_2 \otimes \mathcal{D}_{18})) \\
&P(R_4 \otimes R_{27})Q_{4,27}
\end{aligned}$$

where $P = I_{27} \oplus I_{27} \oplus P_3$ and $P_3 = (I_2 \oplus L_2^4 \oplus L_2^{12} \oplus L_2^{36})L_{27}^{54}$.

3.3.5 The Improved Split-Nesting Algorithm

The split-nesting algorithm combined with the prime power algorithm provides a method for computing any size cyclic convolution. Since the prime power algorithm consists of direct sums of linear convolutions combined with various reductions, and the split-nesting algorithm commutes the direct sums and tensor products, all cyclic convolutions computed via split-nesting become direct sums of reduced tensor products of linear convolutions. It may not be immediately clear as yet, but in fact, any of the linear convolutions and tensor products of linear convolutions can be replaced by other linear convolutions or tensor products of linear convolutions. This is the main idea behind the improved split-nesting algorithm. Simply stated, the improved split-nesting algorithm replaces any linear convolution or tensor product of linear convolutions, with optimal substitutes. Here optimal may mean fastest run-time, fewest operations, etc.

Example 12 *To make this idea more clear, consider example 11 again. One of the components of this algorithm is $\mathcal{C}_2 \otimes \mathcal{D}_{18}$, where $\mathcal{C}_2 = M(x^2+1)\mathcal{L}_2$ and $\mathcal{D}_{18} = M(x^{18}+x^9+1)\mathcal{L}_{18}$, so that this component is really $(M(x^2+1) \otimes M(x^{18}+x^9+1))(\mathcal{L}_2 \otimes \mathcal{L}_{18})$ or more generally a reduction composed with a tensor product of two linear convolutions (e.g. $\mathcal{M}(\mathcal{L}_2 \otimes \mathcal{L}_{18})$).*

It will be shown in chapter 6 that the optimal $\mathcal{L}_2$ is the Toom-Cook linear algorithm tc_2 that requires 6 operations, and that the optimal $\mathcal{L}_{18}$ is $O_{3,2,3} sb_3 \otimes tc_2 \otimes tc_3$, that is obtained by combining the standard algorithm of size 3 with Toom-Cook's of size 2 and 3. It will be shown in chapter 6 that this algorithm requires 366 operations.

Next note that $\mathcal{L}_2 \otimes \mathcal{L}_{18}$ is related to L_{36} since the latter is just $O_{2,18}(\mathcal{L}_2 \otimes \mathcal{L}_{18})$. Because of matrix exchange, the O matrix has no cost, so that the number of operations required for $\mathcal{L}_2 \otimes \mathcal{L}_{18}$, is the same as that of $\mathcal{L}_{36}$.

Table 3.1 below, shows a table of size 36 convolutions built using the methodology discussed in chapter 6. If the optimal $\mathcal{L}_2$ and $\mathcal{L}_{18}$ are chosen, the algorithm for $\mathcal{L}_2 \otimes \mathcal{L}_{18}$ would cost the same as Lin_{36i}, or 1272 operations. However the improved split-nesting algorithm would substitute Lin_{36d}, which could be made equivalent to $\mathcal{L}_2 \otimes \mathcal{L}_{18}$ via the commutation theorem. That is,

$$\begin{aligned}
\mathcal{L}_2 \otimes \mathcal{L}_{18} &= tc_2 \otimes O_{3,2,3}(sb_3 \otimes tc_2 \otimes tc_3) \\
&= (I_3 \otimes O_{3,2,3})(tc_2 \otimes sb_3 \otimes tc_2 \otimes tc_3) \qquad (3.5) \\
&= (I_3 \otimes O_{3,2,3})(L_3^{15}(sb_3 \otimes tc_2)L_3^6 \otimes tc_2 \otimes tc_3) \qquad (3.6)
\end{aligned}$$

Note that (3.5) is related to Lin_{36i} requiring 1272 operations, and that (3.6) is related to Lin_{36d} requiring only 1092 operations. Thus the cost of the size 108 cyclic convolution can be reduced by at least 180 operations via the improved split-nesting algorithm.

In Chapter 4 an infrastructure is discussed that automates the implementation of the algorithms discussed in this chapter.

Table 3.1: Operation Counts for Linear Convolution

Method	B Adds	B Muls	A^t Adds	A^t Muls	Diag Muls	Total Ops
$Lin_{36a} = O_{3,3,2,2}(sb_3 \otimes sb_3 \otimes tc_2 \otimes tc_2)$	45	0	738	0	729	1512
$Lin_{36b} = O_{3,2,3,2}(sb_3 \otimes tc_2 \otimes sb_3 \otimes tc_2)$	99	0	792	0	729	1620
$Lin_{36c} = O_{3,2,2,3}(sb_3 \otimes tc_2 \otimes tc_2 \otimes sb_3)$	135	0	828	0	729	1692
$Lin_{36d} = O_{3,2,2,3}(sb_3 \otimes tc_2 \otimes tc_2 \otimes tc_3)$	159	0	528	0	405	1092
$Lin_{36e} = O_{3,2,3,2}(sb_3 \otimes tc_2 \otimes tc_3 \otimes tc_2)$	189	0	558	0	405	1152
$Lin_{36f} = O_{3,3,2,2}(sb_3 \otimes tc_3 \otimes tc_2 \otimes tc_2)$	234	0	603	0	405	1242
$Lin_{36g} = O_{2,3,3,2}(tc_2 \otimes sb_3 \otimes sb_3 \otimes tc_2)$	261	0	954	0	729	1944
$Lin_{36h} = O_{2,3,2,3}(tc_2 \otimes sb_3 \otimes tc_2 \otimes sb_3)$	297	0	990	0	729	2016
$Lin_{36i} = O_{2,3,2,3}(tc_2 \otimes sb_3 \otimes tc_2 \otimes tc_3)$	249	0	618	0	405	1272
$Lin_{36j} = O_{2,3,3,2}(tc_2 \otimes sb_3 \otimes tc_3 \otimes tc_2)$	279	0	648	0	405	1332
$Lin_{36k} = O_{2,2,3,3}(tc_2 \otimes tc_2 \otimes sb_3 \otimes sb_3)$	405	0	1098	0	729	2232
$Lin_{36l} = O_{2,2,3,3}(tc_2 \otimes tc_2 \otimes sb_3 \otimes tc_3)$	309	0	678	0	405	1392
$Lin_{36m} = O_{2,2,3,3}(tc_2 \otimes tc_2 \otimes tc_3 \otimes sb_3)$	477	0	846	0	405	1728
$Lin_{36n} = O_{2,2,3,3}(tc_2 \otimes tc_2 \otimes tc_3 \otimes tc_3)$	349	0	538	0	225	1112
$Lin_{36o} = O_{2,3,3,2}(tc_2 \otimes tc_3 \otimes sb_3 \otimes tc_2)$	531	0	900	0	405	1836
$Lin_{36p} = O_{2,3,2,3}(tc_2 \otimes tc_3 \otimes tc_2 \otimes sb_3)$	567	0	936	0	405	1908
$Lin_{36q} = O_{2,3,2,3}(tc_2 \otimes tc_3 \otimes tc_2 \otimes tc_3)$	399	0	588	0	225	1212
$Lin_{36r} = O_{2,3,3,2}(tc_2 \otimes tc_3 \otimes tc_3 \otimes tc_2)$	429	0	618	0	225	1272
$Lin_{36s} = O_{3,3,2,2}(tc_3 \otimes sb_3 \otimes tc_2 \otimes tc_2)$	612	0	981	0	405	1998
$Lin_{36t} = O_{3,2,3,2}(tc_3 \otimes tc_2 \otimes sb_3 \otimes tc_2)$	666	0	1035	0	405	2106
$Lin_{36u} = O_{3,2,2,3}(tc_3 \otimes tc_2 \otimes tc_2 \otimes sb_3)$	702	0	1071	0	405	2178
$Lin_{36v} = O_{3,2,2,3}(tc_3 \otimes tc_2 \otimes tc_2 \otimes tc_3)$	474	0	663	0	225	1362
$Lin_{36w} = O_{3,2,3,2}(tc_3 \otimes tc_2 \otimes tc_3 \otimes tc_2)$	504	0	693	0	225	1422
$Lin_{36x} = O_{3,3,2,2}(tc_3 \otimes tc_3 \otimes tc_2 \otimes tc_2)$	549	0	738	0	225	1512