Chapter 8: Fast Convolution

Keshab K. Parhi

Chapter 8 Fast Convolution

- Introduction
- Cook-Toom Algorithm and Modified Cook-Toom Algorithm
- Winograd Algorithm and Modified Winograd Algorithm
- Iterated Convolution
- Cyclic Convolution
- Design of Fast Convolution Algorithm by Inspection

Introduction

- **Fast Convolution**: implementation of convolution algorithm using fewer multiplication operations by **algorithmic strength reduction**
- **Algorithmic Strength Reduction:** Number of strong operations (such as multiplication operations) is reduced at the expense of an increase in the number of weak operations (such as addition operations). These are best suited for implementation using either programmable or dedicated hardware
- **Example:** Reducing the multiplication complexity in complex number multiplication:
  - Assume (a+jb)(c+dj)=e+jf, it can be expressed using the matrix form, which requires 4 multiplications and 2 additions:

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

  - However, the number of multiplications can be reduced to 3 at the expense of 3 extra additions by using:

$$\begin{cases} ac - bd = a(c-d) + d(a-b) \\ ad + bc = b(c+d) + d(a-b) \end{cases}$$

- Rewrite it into matrix form, its coefficient matrix can be decomposed as the product of a 2X3(**C**), a 3X3(**H**)and a 3X2(**D**) matrix:

$$s=\begin{bmatrix} e \\ f \end{bmatrix}=\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}\cdot\begin{bmatrix} c-d & 0 & 0 \\ 0 & c+d & 0 \\ 0 & 0 & d \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}\cdot\begin{bmatrix} a \\ b \end{bmatrix}=C\cdot H\cdot D\cdot x$$

  - Where C is a post-addition matrix (requires 2 additions), D is a pre-addition matrix (requires 1 addition), and H is a diagonal matrix (requires 2 additions to get its diagonal elements)
- So, the arithmetic complexity is reduced to 3 multiplications and 3 additions (not including the additions in **H** matrix)

- In this chapter we will discuss two well-known approaches to the design of fast short-length convolution algorithms: the **Cook-Toom algorithm** (based on **Lagrange Interpolation**) and the **Winograd Algorithm** (based on the **Chinese remainder theorem**)

Cook-Toom Algorithm

- A linear convolution algorithm for polynomial multiplication based on the ***Lagrange Interpolation Theorem***
- Lagrange Interpolation Theorem:

Let $b_0, \ldots, b_n$ be a set of $n+1$ distinct points, and let $f(b_i)$, for i = 0, 1, …, n be given. There is exactly one polynomial $f(p)$ of degree n or less that has value $f(b_i)$ when evaluated at b_i for i = 0, 1, …, n. It is given by:

$$f(p) = \sum_{i=0}^{n} f(b_i) \frac{\prod_{j \neq i} (p - b_j)}{\prod_{j \neq i} (b_i - b_j)}$$

- The application of Lagrange interpolation theorem into linear convolution

  Consider an N-point sequence $h = \{h_0, h_1, ..., h_{N-1}\}$ and an L-point sequence $x = \{x_0, x_1, ..., x_{L-1}\}$. The linear convolution of h and x can be expressed in terms of polynomial multiplication as follows: $s(p) = h(p) \cdot x(p)$ where

  $$h(p) = h_{N-1} p^{N-1} + ... + h_1 p + h_0$$

  $$x(p) = x_{L-1} p^{L-1} + ... + x_1 p + x_0$$

  $$s(p) = s_{L+N-2} p^{L+N-2} + ... + s_1 p + s_0$$

  The output polynomial $s(p)$ has degree $L + N - 2$ and has $L + N - 1$ different points.

- (continued)

$s(p)$ can be uniquely determined by its values at $L+N-1$ different points. Let $\{b_0, b_1, ..., b_{L+N-2}\}$ be $L+N-1$ different real numbers. If $s(b_i)$ for $i = \{0,1,..., L+N-2\}$ are known, then $s(p)$ can be computed using the Lagrange interpolation theorem as:

$$s(p) = \sum_{i=0}^{L+N-2} s(b_i) \frac{\prod_{j \neq i} (p - b_j)}{\prod_{j \neq i} (b_i - b_j)}$$

It can be proved that this equation is the unique solution to compute linear convolution for $s(p)$ given the values of $s(b_i)$, for $i = \{0,1,..., L+N-2\}$.

- Cook-Toom Algorithm (Algorithm Description)

1. **Choose** $L+N-1$ **different real numbers** $b_0, b_1, \cdots b_{L+N-2}$
2. **Compute** $h(b_i)$ **and** $x(b_i)$ **, for** $i = \{0,1,\cdots, L+N-2\}$
3. **Compute** $s(b_i) = h(b_i) \cdot x(b_i)$ **, for** $i = \{0,1,\cdots, L+N-2\}$
4. **Compute** $s(p)$ **by using** $$s(p) = \sum_{i=0}^{L+N-2} s(b_i) \frac{\prod_{j \neq i} (p - b_j)}{\prod_{j \neq i} (b_i - b_j)}$$

- Algorithm Complexity
  - The goal of the fast-convolution algorithm is to reduce the multiplication complexity. So, if β_i `s (i=0,1,…,L+N-2) are chosen properly, the computation in step-2 involves some additions and multiplications by small constants
  - The multiplications are only used in step-3 to compute $s(\beta_i)$. So, only L+N-1 multiplications are needed

- By Cook-Toom algorithm, the number of multiplications is reduced from O(LN) to L+N-1 at the expense of an increase in the number of additions
- An adder has much less area and computation time than a multiplier. So, the Cook-Toom algorithm can lead to large savings in hardware (VLSI) complexity and generate computationally efficient implementation

- Example-1: (Example 8.2.1, p.230) Construct a 2X2 convolution algorithm using Cook-Toom algorithm with $\beta=\{0,1,-1\}$
  - Write 2X2 convolution in polynomial multiplication form as s(p)=h(p)x(p), where $h(p) = h_0 + h_1 p \quad x(p) = x_0 + x_1 p$

$$s(p) = s_0 + s_1 p + s_2 p^2$$

  - Direct implementation, which requires 4 multiplications and 1 additions, can be expressed in matrix form as follows:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

- Example-1 (continued)
  - Next we use C-T algorithm to get an efficient convolution implementation with reduced multiplication number

$$b_0 = 0, \quad h(b_0) = h_0, \quad x(b_0) = x_0$$
$$b_1 = 1, \quad h(b_1) = h_0 + h_1, \quad x(b_1) = x_0 + x_1$$
$$b_2 = 2, \quad h(b_2) = h_0 - h_1, \quad x(b_2) = x_0 - x_1$$

  - Then, $s(\beta_0)$, $s(\beta_1)$, and $s(\beta_2)$ are calculated, by using 3 multiplications, as

$$s(b_0) = h(b_0)x(b_0) \quad s(b_1) = h(b_1)x(b_1) \quad s(b_2) = h(b_2)x(b_2)$$

  - From the Lagrange Interpolation theorem, we get:

$$s(p) = s(b_0)\frac{(p-b_1)(p-b_2)}{(b_0-b_1)(b_0-b_2)} + s(b_1)\frac{(p-b_0)(p-b_1)}{(b_1-b_0)(b_1-b_2)}$$
$$+ s(b\ 2)\frac{(p-b_0)(p-b_1)}{(b_2-b_0)(b_2-b_1)}$$
$$= s(b_0) + p\left(\frac{s(b_1)-s(b_2)}{2}\right) + p^2\left(-s(b_0) + \frac{s(b_1)+s(b_2)}{2}\right)$$
$$= s_0 + ps_1 + p^2 s_2$$

- Example-1 (continued)
  - The preceding computation leads to the following matrix form

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} s(b_0) \\ s(b_1)/2 \\ s(b_2)/2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} h(b_0) & 0 & 0 \\ 0 & h(b_1)/2 & 0 \\ 0 & 0 & h(b_2)/2 \end{bmatrix} \cdot \begin{bmatrix} x(b_0) \\ x(b_1) \\ x(b_2) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 \\ 0 & (h_0 + h_1)/2 & 0 \\ 0 & 0 & (h_0 - h_1)/2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

  - The computation is carried out as follows **(5 additions, 3 multiplications)**

1. $H_0 = h_0, \quad H_1 = \dfrac{h_0 + h_1}{2}, \quad H_2 = \dfrac{h_0 - h_1}{2}$ **(pre-computed)**
2. $X_0 = x_0, \quad X_1 = x_0 + x_1, \quad X_2 = x_0 - x_1$
3. $S_0 = H_0 X_0, \quad S_1 = H_1 X_1, \quad S_2 = H_2 X_2$
4. $s_0 = S_0, \quad s_1 = S_1 - S_2, \quad s_2 = -S_0 + S_1 + S_2$

- (Continued): Therefore, this algorithm needs 3 multiplications and 5 additions (ignoring the additions in the pre-computation), i.e., the number of multiplications is reduced by 1 at the expense of 4 extra additions
- **Example-2**, please see **Example 8.2.2 of Textbook (p.231)**

- Comments
  - Some additions in the **preaddition** or **postaddition** matrices can be **shared**. So, when we count the number of additions, we only count one instead of two or three.
  - If we take h_0, h_1 as the FIR filter coefficients and take x_0, x_1 as the signal (data) sequence, then the terms **H_0, H_1 need not be recomputed** each time the filter is used. They can be precomputed once offline and stored. So, we **ignore** these computations when counting the number of operations
  - From Example-1, We can understand the Cook-Toom algorithm as a matrix decomposition. In general, a convolution can be expressed in matrix-vector forms as $\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$ or $s = T \cdot x$

– Generally, the equation can be expressed as

$$s = T \cdot x = C \cdot H \cdot D \cdot x$$

  - Where C is the postaddition matrix, D is the preaddition matrix, and H is a diagonal matrix with H_i, i = 0, 1, …, L+N-2 on the main diagonal.

– Since **T=CHD**, it implies that the Cook-Toom algorithm provides a way to factorize the convolution matrix T into multiplication of 1 postaddition matrix **C**, 1 diagonal matrix **H** and 1 preaddition matrix **D**, such that the total number of multiplications is determined only by the non-zero elements on the main diagonal of the diagonal matrix **H**

– Although the number of multiplications is reduced, the number of additions has increased. The Cook-Toom algorithm can be modified in order to further reduce the number of additions

Modified Cook-Toom Algorithm

- The Cook-Toom algorithm is used to further reduce the number of addition operations in linear convolutions

  Define $s'(p) = s(p) - S_{L+N-2} p^{L+N-2}$. Notice that the degree of $s(p)$ is $L+N-2$ and S_{L+N-2} is its highest order coefficient. Therefore the degree of $s'(p)$ is $L+N-3$.

- Now consider the modified Cook-Toom Algorithm

- Modified Cook-Toom Algorithm

1. **Choose** $L+N-2$ **different real numbers** $b_0, b_1, \cdots b_{L+N-3}$
2. **Compute** $h(b_i)$ **and** $x(b_i)$ **, for** $i = \{0,1,\cdots, L+N-3\}$
3. **Compute** $s(b_i) = h(b_i) \cdot x(b_i)$ **, for** $i = \{0,1,\cdots, L+N-3\}$
4. **Compute** $s'(b_i) = s(b_i) - s_{L+N-2} b_i^{L+N-2}$ **, for** $i = \{0,1,\cdots, L+N-3\}$
5. **Compute** $s'(p)$ **by using** $$s'(p) = \sum_{i=0}^{L+N-2} s'(b_i) \frac{\prod_{j \neq i} (p - b_j)}{\prod_{j \neq i} (b_i - b_j)}$$
6. **Compute** $s(p) = s'(p) + s_{L+N-2} p^{L+N-2}$

- Example-3 (Example 8.2.3, p.234) Derive a 2X2 convolution algorithm using the modified Cook-Toom algorithm with β={0,-1}

  Consider the Lagrange interpolation for $s'(p) = s(p) - h_1 x_1 p^2$ at $\{b_0 = 0, b_1 = -1\}$.

  First, find $s'(b_i) = h(b_i)x(b_i) - h_1 x_1 b_i^2$

  $$b_0 = 0, \quad h(b_0) = h_0, \quad x(b_0) = x_0$$

  $$b_1 = -1, \quad h(b_1) = h_0 - h_1, \quad x(b_1) = x_0 - x_1$$

  – and

  $$s'(b_0) = h(b_0)x(b_0) - h_1 x_1 b_0^2 = h_0 x_0$$

  $$s'(b_1) = h(b_1)x(b_1) - h_1 x_1 b_1^2 = (h_0 - h_1)(x_0 - x_1) - h_1 x_1$$

  - Which requires 2 multiplications (not counting the $h_1 x_1$ multiplication)

  – Apply the Lagrange interpolation algorithm, we get:

  $$s'(p) = s'(b_0)\frac{(p - b_1)}{(b_0 - b_1)} + s'(b_1)\frac{(p - b_0)}{(b_1 - b_0)}$$

  $$= s'(b_0) + p(s'(b_0) - s'(b_1))$$

- Example-3 (cont'd)
  - Therefore, $s(p) = s'(p) + h_1 x_1 p^2 = s_0 + s_1 p + s_2 p^2$
  - Finally, we have the matrix-form expression:
  - Notice that
$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s'(b_0) \\ s'(b_1) \\ h_1 x_1 \end{bmatrix}$$
  - Therefore:
$$\begin{bmatrix} s'(b_0) \\ s'(b_1) \\ h_1 x_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s(b_0) \\ s(b_1) \\ h_1 x_1 \end{bmatrix}$$
$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s(b_0) \\ s(b_1) \\ h_1 x_1 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_0 - h_1 & 0 \\ 0 & 0 & h_1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

- Example-3 (cont'd)
  - The computation is carried out as the follows:
    1. $H_0 = h_0, \quad H_1 = h_0 - h_1, \quad H_2 = h_1$ **(pre-computed)**
    2. $X_0 = x_0, \quad X_1 = x_0 - x_1, \quad X_2 = x_1$
    3. $S_0 = H_0 X_0, \quad S_1 = H_1 X_1, \quad S_2 = H_2 X_2$
    4. $s_0 = S_0, \quad s_1 = S_0 - S_1 + S_2, \quad s_2 = S_2$
  - The total number of operations are 3 multiplications and 3 additions. Compared with the convolution algorithm in Example-1, the number of addition operations has been reduced by 2 while the number of multiplications remains the same.
- **Example-4 (Example 8.2.4, p. 236 of Textbook)**
- **Conclusion:** The Cook-Toom Algorithm is efficient as measured by the number of multiplications. However, as the size of the problem increases, it is not efficient because the number of additions increases greatly if β takes values other than $\{0, \pm 1, \pm 2, \pm 4\}$. This may result in complicated pre-addition and post-addition matrices. For large-size problems, the Winograd algorithm is more efficient.

Winograd Algorithm

- The Winograd short convolution algorithm: based on the CRT (Chinese Remainder Theorem) ---It's possible to uniquely determine a nonnegative integer given only its remainder with respect to the given moduli, provided that the moduli are relatively prime and the integer is known to be smaller than the product of the moduli
- Theorem: CRT for Integers

Given $c_i = R_{m_i}[c]$ **(represents the remainder when** c **is divided by** m_i **), for** $i = 0,1,\ldots,k$, **where** m_i **are moduli and are relatively prime, then**

$$c = \left(\sum_{i=0}^{k} c_i N_i M_i \right) \bmod M$$

, **where** $M = \prod_{i=0}^{k} m_i$, $M_i = M / m_i$, **and** N_i **is the solution of** $N_i M_i + n_i m_i = GCD(M_i, m_i) = 1$, **provided that** $0 \le c < M$

- **Theorem: CRT for Polynomials**

  Given $c^{(i)}(p) = R_{m^{(i)}}(p)[c(p)]$, for i=0, 1, …,k, where $m^{(i)}(p)$ are relatively prime, then $c(p) = \left(\sum_{i=0}^{k} c^{(i)}(p) N^{(i)}(p) M^{(i)}(p) \right) \bmod M(p)$, where $M(p) = \prod_{i=0}^{k} m^{(i)}(p)$, $M^{(i)}(p) = M(p) / m^{(i)}(p)$, and $N^{(i)}(p)$ is the solution of $N^{(i)}(p) M^{(i)}(p) + n^{(i)}(p) m^{(i)}(p) = GCD(M^{(i)}(p), m^{(i)}(p)) = 1$ Provided that the degree of $c(p)$ is less than the degree of $M(p)$

- **Example-5** (Example 8.3.1, p.239)**: using the CRT for integer**, Choose moduli m_0=3, m_1=4, m_2=5. Then $M = m_0 m_1 m_2 = 60$, and $M_i = M / m_i$. Then:

$$m_0 = 3, \quad M_0 = 20, \quad (-1)20 + 7(3) = 1$$
$$m_1 = 4, \quad M_1 = 15, \quad (-1)15 + (4)4 = 1$$
$$m_2 = 5, \quad M_2 = 12, \quad (-2)12 + (5)5 = 1$$

  – where N_i and n_i are obtained using the Euclidean GCD algorithm. Given that the integer c satisfying $0 \le c < M$, let $c_i = R_{m_i}[c]$.

- **Example-5 (cont'd)**
  - The integer c can be calculated as

$$c = \left(\sum_{i=0}^{k} c_i N_i M_i \right) \bmod M = (-20 * c_0 - 15 * c_1 - 24 * c_2) \bmod 60$$

  - For c=17, $c_0 = R_3(17) = 2, \quad c_1 = R_4(17) = 1, \quad c_2 = R_5(17) = 2$

$$c = (-20 * 2 - 15 * 1 - 24 * 2) \bmod 60 = (-103) \bmod 60 = 17$$

- **CRT for polynomials:** The remainder of a polynomial with regard to modulus $p^i + f(p)$, where $\deg(f(p)) \le i - 1$, can be evaluated by substituting p^i by $-f(p)$ in the polynomial
- **Example-6** (Example 8.3.2, pp239)

$$(a). \quad R_{x+2}\left[5x^2 + 3x + 5\right] = 5(-2)^2 + 3(-2) + 5 = 19$$

$$(b). \quad R_{x^2+2}\left[5x^2 + 3x + 5\right] = 5(-2) + 3x + 5 = 3x - 5$$

$$(c). \quad R_{x^2+x+2}\left[5x^2 + 3x + 5\right] = 5(-x - 2) + 3x + 5 = -2x - 5$$

- Winograd Algorithm
  - 1. Choose a polynomial $m(p)$ with degree higher than the degree of $h(p)x(p)$ and factor it into k+1 relatively prime polynomials with real coefficients, i.e., $m(p) = m^{(0)}(p)m^{(1)}(p)\cdots m^{(k)}(p)$
  - 2. Let $M^{(i)}(p) = m(p)/m^{(i)}(p)$. Use the Euclidean GCD algorithm to solve $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1$ for $N^{(i)}(p)$.
  - 3. Compute: $h^{(i)}(p) = h(p) \bmod m^{(i)}(p), \qquad x^{(i)}(p) = x(p) \bmod m^{(i)}(p)$ $for \quad i = 0,1,\cdots,k$
  - 4. Compute: $s^{(i)}(p) = h^{(i)}(p)x^{(i)}(p) \bmod m^{(i)}(p), \qquad for \quad i = 0,1,\cdots,k$
  - 5. Compute $s(p)$ by using:

$$s(p) = \sum_{i=0}^{k} s^{(i)}(p)N^{(i)}(p)M^{(i)}(p) \bmod m^{(i)}(p)$$

- Example-7 (Example 8.3.3, p.240) Consider a 2X3 linear convolution as in Example 8.2.2. Construct an efficient realization using Winograd algorithm with $m(p) = p(p-1)(p^2+1)$
  - Let: $m^{(0)}(p) = p, \quad m^{(1)}(p) = p-1, \quad m^{(2)}(p) = p^2+1$
  - Construct the following table using the relationships $M^{(i)}(p) = m(p)/m^{(i)}(p)$ and $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1 \quad for \quad i = 0,1,2$

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	p	$p^3 - p^2 + p - 1$	$p^2 - p + 1$	-1
1	$p-1$	$p^3 + p$	$-\frac{1}{2}\left(p^2 + p + 2\right)$	$\frac{1}{2}$
2	p^2+1	$p^2 - p$	$-\frac{1}{2}(p-2)$	$\frac{1}{2}(p-1)$

  - Compute residues from $h(p) = h_0 + h_1 p, \quad x(p) = x_0 + x_1 p + x_2 p^2$:

$$h^{(0)}(p) = h_0, \qquad x^{(0)}(p) = x_0$$
$$h^{(1)}(p) = h_0 + h_1, \qquad x^{(1)}(p) = x_0 + x_1 + x_2$$
$$h^{(2)}(p) = h_0 + h_1 p, \quad x_{(2)}(p) = (x_0 - x_2) + x_1 p$$

- Example-7 (cont'd)

$$s^{(0)}(p) = h_0 x_0 = s_0^{(0)}, \quad s^{(1)}(p) = (h_0 + h_1)(x_0 + x_1 + x_2) = s_0^{(1)}$$

$$s^{(2)}(p) = (h_0 + h_1 p)((x_0 - x_2) + x_1 p) \bmod (p^2 + 1)$$

$$= h_0(x_0 - x_2) - h_1 x_1 + (h_0 x_1 + h_1(x_0 - x_2))\, p = s_0^{(2)} + s_1^{(2)} p$$

  - Notice, we need 1 multiplication for $s^{(0)}(p)$, 1 for $s^{(1)}(p)$, and 4 for $s^{(2)}(p)$
  - However it can be further reduced to 3 multiplications as shown below:

$$\begin{bmatrix} s_0^{(2)} \\ s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_0 - h_1 & 0 \\ 0 & 0 & h_0 + h_1 \end{bmatrix} \cdot \begin{bmatrix} x_0 + x_1 - x_2 \\ x_0 - x_2 \\ x_1 \end{bmatrix}$$

  - Then:

$$s(p) = \sum_{i=0}^{2} s^{(i)}(p) N^{(i)}(p) M^{(i)}(p) \bmod m^{(i)}(p)$$

$$= \left[-s^{(0)}(p)(p^3 - p^2 + p - 1) + \frac{S^{(1)}(p)}{2}(p^3 + p) + \frac{S^{(2)}(p)}{2}(p^3 - 2p^2 + p) \right]$$

$$\bmod (p^4 - p^3 + p^2 - p)$$

- Example-7 (cont'd)
  - Substitute $s^{(0)}(p)$, $s^{(1)}(p)$, $s^{(2)}(p)$ into $s(p)$ to obtain the following table

p^0	p^1	p^2	p^3
$s_0^{(0)}$	$-s_0^{(0)}$	$s_0^{(0)}$	$-s_0^{(0)}$
0	$\frac{1}{2}s_0^{(1)}$	0	$\frac{1}{2}s_0^{(1)}$
0	$\frac{1}{2}s_0^{(2)}$	$-s_0^{(2)}$	$\frac{1}{2}s_0^{(2)}$
0	$\frac{1}{2}s_1^{(2)}$	0	$-\frac{1}{2}s_1^{(2)}$

  - Therefore, we have

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 0 & -2 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} s_0^{(0)} \\ \frac{1}{2}s_0^{(1)} \\ \frac{1}{2}s_0^{(2)} \\ \frac{1}{2}s_1^{(2)} \end{bmatrix}$$

- Example-7 (cont'd)
  - Notice that

$$\begin{bmatrix} s_0^{(0)} \\ \frac{1}{2} s_0^{(1)} \\ \frac{1}{2} s_0^{(2)} \\ \frac{1}{2} s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{h_1-h_0}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1}{2} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_0 + x_1 + x_2 \\ x_0 + x_1 - x_2 \\ x_0 - x_2 \\ x_1 \end{bmatrix}$$

  - So, finally we have:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 1 & -1 \\ 1 & 0 & -2 & 0 & 2 \\ -1 & 0 & 0 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{h_1-h_0}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1}{2} \end{bmatrix}$$

$$\cdot \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

- Example-7 (cont'd)
  - In this example, the Winograd convolution algorithm requires 5 multiplications and 11 additions compared with 6 multiplications and 2 additions for direct implementation
- Notes:
  - The number of multiplications in Winograd algorithm is highly dependent on the degree of each $m^{(i)}(p)$. Therefore, the degree of m(p) should be as small as possible.
  - More efficient form (or a modified version) of the Winograd algorithm can be obtained by letting deg[m(p)]=deg[s(p)] and applying the CRT to

$$s'(p) = s(p) - h_{N-1} x_{L-1} m(p)$$

Modified Winograd Algorithm

- 1. Choose a polynomial $m(p)$ with degree equal to the degree of $s(p)$ and factor it into k+1 relatively prime polynomials with real coefficients, i.e., $m(p) = m^{(0)}(p)m^{(1)}(p)\cdots m^{(k)}(p)$
- 2. Let $M^{(i)}(p) = m(p)/m^{(i)}(p)$, use the Euclidean GCD algorithm to solve $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1$ for $N^{(i)}(p)$.
- 3. Compute: $h^{(i)}(p) = h(p) \bmod m^{(i)}(p), \qquad x^{(i)}(p) = x(p) \bmod m^{(i)}(p)$ $for \quad i = 0,1,\cdots,k$
- 4. Compute: $s'^{(i)}(p) = h^{(i)}(p)x^{(i)}(p) \bmod m^{(i)}(p), \qquad for \quad i = 0,1,\cdots,k$
- 5. Compute $s'(p)$ by using:
$$s'(p) = \sum_{i=0}^{k} s'^{(i)}(p)N^{(i)}(p)M^{(i)}(p) \bmod m^{(i)}(p)$$
- 6. Compute $s(p) = s'(p) + h_{N-1}x_{L-1}m(p)$

- Example-8 (Example 8.3.4, p.243): Construct a 2X3 convolution algorithm using modified Winograd algorithm with m(p)=p(p-1)(p+1)
  - Let $m^{(0)}(p) = p, \quad m^{(1)}(p) = p-1, \quad m^{(2)}(p) = p+1$
  - Construct the following table using the relationships $M^{(i)}(p) = m(p)/m^{(i)}(p)$ and $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1$

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	p	p^2-1	p	-1
1	$p-1$	p^2+p	$-\frac{1}{2}(p+2)$	$\frac{1}{2}$
2	$p+1$	p^2-p	$-\frac{1}{2}(p-2)$	$\frac{1}{2}$

  - Compute residues from $h(p) = h_0 + h_1 p, \quad x(p) = x_0 + x_1 p + x_2 p^2$:

$$h^{(0)}(p) = h_0, \qquad x^{(0)}(p) = x_0$$
$$h^{(1)}(p) = h_0 + h_1, \quad x^{(1)}(p) = x_0 + x_1 + x_2$$
$$h^{(2)}(p) = h_0 - h_1, \quad x_{(2)}(p) = x_0 - x_1 + x_2$$
$$s'^{(0)}(p) = h_0 x_0, \quad s'^{(1)}(p) = (h_0 + h_1)(x_0 + x_1 + x_2),$$
$$s'^{(2)}(p) = (h_0 - h_1)(x_0 - x_1 + x_2)$$

- Example-8 (cont'd)
  - Since the degree of $m^{(i)}(p)$ is equal to 1, $s'^{(i)}(p)$ is a polynomial of degree 0 (a constant). Therefore, we have:

$$\begin{aligned} s(p) &= s'(p) + h_1 x_2 m(p) \\ &= \left[-s'^{(0)}(-p^2+1) + \frac{s'^{(1)}}{2}(p^2+p) + \frac{s'^{(2)}}{2}(p^2-p) + h_1 x_2 (p^3-p) \right] \\ &= s'^{(0)} + p\left(\frac{s'(1)}{2} - \frac{s'(2)}{2} - h_1 x_2\right) + p^2\left(-s'^{(0)} + \frac{s'^{(1)}}{2} + \frac{s'^{(2)}}{2}\right) + p^3(h_1 x_2) \end{aligned}$$

  - The algorithm can be written in matrix form as:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s'^{(0)} \\ \frac{s'^{(1)}}{2} \\ \frac{s'^{(2)}}{2} \\ h_1 x_2 \end{bmatrix}$$

- Example-8 (cont'd)
  - (matrix form)

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & \frac{h_0 + h_1}{2} & 0 & 0 \\ 0 & 0 & \frac{h_0 - h_1}{2} & 0 \\ 0 & 0 & 0 & h_1 \end{bmatrix}$$

$$\cdot \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

  - Conclusion: this algorithm requires 4 multiplications and 7 additions

Iterated Convolution

- I**terated convolution algorithm:** makes use of efficient short-length convolution algorithms **iteratively** to build long convolutions
- Does not achieve minimal multiplication complexity, but achieves a **good balance** between multiplications and addition complexity
- Iterated Convolution Algorithm (Description)
  - 1. Decompose the long convolution into several levels of short convolutions
  - 2. Construct fast convolution algorithms for short convolutions
  - 3. Use the short convolution algorithms to iteratively (hierarchically) implement the long convolution
  - **Note**: the order of short convolutions in the decomposition affects the complexity of the derived long convolution

- **Example-9** (Example 8.4.1, pp.245): Construct a 4X4 linear convolution algorithm using 2X2 short convolution
  - Let $h(p) = h_0 + h_1 p + h_2 p^2 + h_3 p^3, \quad x(p) = x_0 + x_1 p + x_2 p^2 + x_3 p^3$ and $s(p) = h(p)x(p)$
  - First, we need to decompose the 4X4 convolution into a 2X2 convolution
  - Define $h'_0(p) = h_0 + h_1 p, \quad h'_1(p) = h_2 + h_3 p$

    $x'_0(p) = x_0 + x_1 p, \quad x'_1(p) = x_2 + x_3 p$
  - Then, we have:

$$\Rightarrow \quad h(p) = h'_0(p) + h'_1(p)p^2, \quad i.e., h(p) = h(p,q) = h'_0(p) + h'_1(p)q$$
$$x(p) = x'_0(p) + x'_1(p)p^2, \quad i.e., x(p) = x(p,q) = x'_0(p) + x'_1(p)q$$

$$s(p) = h(p)x(p) = h(p,q)x(p,q)$$
$$\Rightarrow \quad = [h'_0(p) + h'_1(p)q] \cdot [x'_0(p) + x'_1(p)q]$$
$$= h'_0(p)x'_0(p) + [h'_0(p)x'_1(p) + h'_1(p)x'_0(p)]q + h'_1(p)x'_1(p)q^2$$
$$= s'_0(p) + s'_1(p)q + s'_2(p)q^2 = s(p,q)$$

- Example-9 (cont'd)
  - Therefore, the 4X4 convolution is decomposed into two levels of nested 2X2 convolutions
  - Let us start from the first convolution $s'_0(p) = h'_0(p) \cdot x'_0(p)$, we have:

$$h'_0(p) \cdot x'_0(p) \equiv h'_0 \cdot x'_0 = (h_0 + h_1 p) \cdot (x_0 + x_1 p)$$
$$= h_0 x_0 + h_1 x_1 p^2 + p[(h_0 + h_1) \cdot (x_0 + x_1) - h_0 x_0 - h_1 x_1]$$

  - We have the following expression for the third convolution:

$$s'_2(p) = h'_1(p) \cdot x'_1(p) \equiv h'_1 \cdot x'_1 = (h_2 + h_3 p) \cdot (x_2 + x_3 p)$$
$$= h_2 x_2 + h_3 x_3 p^2 + p[(h_2 + h_3) \cdot (x_2 + x_3) - h_2 x_2 - h_3 x_3]$$

  - For the second convolution, we get the following expression:

$$s'_1(p) = h'_0(p) \cdot x'_1(p) + h'_1(p) \cdot x'_0(p) \equiv h'_0 \cdot x'_1 + h'_1 \cdot x'_0$$
$$= [(h'_0 + h'_1) \cdot (x'_0 + x'_1) - h'_0 \cdot x'_0 - h'_1 \cdot x'_1]$$

—: multiplication ▼: addition

- Example-9 (Cont'd)
  - For $[(h'_0+h'_1)\cdot(x'_0+x'_1)]$, we have the following expression:

$$(h'_0+h'_1)\cdot(x'_0+x'_1)=[(h_0+h_2)+p(h_1+h_3)]\cdot[(x_0+x_2)+p(x_1+x_3)]$$
$$=(h_0+h_2)\cdot(x_0+x_2)+p^2(h_1+h_3)\cdot(x_1+x_3)$$
$$+p[(h_0+h_1+h_2+h_3)\cdot(x_0+x_1+x_2+x_3)$$
$$-(h_0+h_2)\cdot(x_0+x_2)-(h_1+h_3)\cdot(x_1+x_3)]$$

This requires 9 multiplications and 11 additions

  - If we rewrite the three convolutions as the following expressions, then we can get the following table (see the next page):

$$h'_0\,x'_0\equiv a_1+pa_2+p^2a_3$$
$$h'_1\,x'_1\equiv b_1+pb_2+p^2b_3$$
$$(h'_0+h'_1)\cdot(x'_0+x'_1)\equiv c_1+pc_2+p^2c_3$$

- Example-9 (cont'd)

p^0	p^1	p^2	p^3	p^4	p^5	p^6
a_1	a_2	a_3		b_1	b_2	b_3
		c_1	c_2	c_3		
		$-b_1$	$-b_2$	$-b_3$		
		$-a_1$	$-a_2$	$-a_3$		

Total 8 additions here

  - Therefore, the total number of operations used in this 4X4 iterated convolution algorithm is 9 multiplications and 19 additions

Cyclic Convolution

- Cyclic convolution: also known as circular convolution
- Let the filter coefficients be $h = \{h_0, h_1, \cdots, h_{n-1}\}$, and the data sequence be $x = \{x_0, x_1, \cdots, x_{n-1}\}$.
  - The cyclic convolution can be expressed as
    $$s(p) = h \mathrm{O}_n x = [h(p) \cdot x(p)] \bmod (p^n - 1)$$
  - The output samples are given by
    $$s_i = \sum_{k=0}^{n-1} h_{((i-k))} x_k, \qquad i = 0,1,\cdots,n-1$$
    - where $((i-k))$ denotes $(i-k) \bmod n$
- The cyclic convolution can be computed as a linear convolution reduced by modulo $p^n - 1$. (Notice that there are 2n-1 different output samples for this linear convolution). Alternatively, the cyclic convolution can be computed using CRT with $m(p) = p^n - 1$, which is much simpler.

- Example-10 (Example 8.5.1, p.246) Construct a 4X4 cyclic convolution algorithm using CRT with $m(p) = p^4 - 1 = (p-1)(p+1)(p^2+1)$
  - Let $h(p) = h_0 + h_1 p + h_2 p^2 + h_3 p^3, \quad x(p) = x_0 + x_1 p + x_2 p^2 + x_3 p^3$
  - Let $m^{(0)}(p) = p - 1, \quad m^{(1)}(p) = p + 1, \quad m^{(2)}(p) = p^2 + 1$
  - Get the following table using the relationships $M^{(i)}(p) = m(p) / m^{(i)}(p)$ and $N^{(i)}(p) M^{(i)}(p) + n^{(i)}(p) m^{(i)}(p) = 1$

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	$p-1$	p^3+p^2+p-1	$-\frac{1}{4}(p^2+2p+3)$	$\frac{1}{4}$
1	$p+1$	p^3-p^2+p-1	$\frac{1}{4}\left(p^2-2p+3\right)$	$-\frac{1}{4}$
2	p^2+1	p^2-1	$\frac{1}{2}$	$-\frac{1}{2}$

  - Compute the residues

$$h^{(0)}(p) = h_0 + h_1 + h_2 + h_3 = h_0^{(0)},$$
$$h^{(1)}(p) = h_0 - h_1 + h_2 - h_3 = h_0^{(1)},$$
$$h^{(2)}(p) = (h_0 - h_2) + (h_1 - h_3)p = h_0^{(2)} + h_1^{(2)} p$$

- Example-10 (cont'd) —: multiplication

$$x^{(0)}(p) = x_0 + x_1 + x_2 + x_3 = x_0^{(0)},$$
$$x^{(1)}(p) = x_0 - x_1 + x_2 - x_3 = x_0^{(1)},$$
$$x^{(2)}(p) = (x_0 - x_2) + (x_1 - x_3)p = x_0^{(2)} + x_1^{(2)} p$$

$\Rightarrow$

$$s^{(0)}(p) = h^{(0)}(p) \cdot x^{(0)}(p) = h_0^{(0)} \cdot x_0^{(0)} = s_0^{(0)},$$
$$s^{(1)}(p) = h^{(1)}(p) \cdot x^{(1)}(p) = h_0^{(1)} \cdot x_0^{(1)} = s_0^{(1)},$$
$$s^{(2)}(p) = s_0^{(2)} + s_1^{(2)} p = [h^{(2)}(p) \cdot x^{(2)}(p)] \bmod (p^2 + 1)$$
$$= \left(h_0^{(2)} \cdot x_0^{(2)} - h_1^{(2)} \cdot x_1^{(2)}\right) + p\left(h_0^{(2)} x_1^{(2)} + h_1^{(2)} x_0^{(2)}\right)$$

  - Since

$$s_0^{(2)} = h_0^{(2)} x_0^{(2)} - h_1^{(2)} x_1^{(2)} = h_0^{(2)}\left(x_0^{(2)} + x_1^{(2)}\right) - \left(h_0^{(2)} + h_1^{(2)}\right) x_1^{(2)},$$
$$s_1^{(2)} = h_0^{(2)} x_1^{(2)} + h_1^{(2)} x_0^{(2)} = h_0^{(2)}\left(x_0^{(2)} + x_1^{(2)}\right) + \left(h_1^{(2)} - h_0^{(2)}\right) x_0^{(2)},$$

  - or in matrix-form

$$\begin{bmatrix} s_0^{(2)} \\ s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} h_0^{(2)} & 0 & 0 \\ 0 & h_1^{(2)} - h_0^{(2)} & 0 \\ 0 & 0 & h_0^{(2)} + h_1^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_0^{(2)} + x_1^{(2)} \\ x_0^{(2)} \\ x_1^{(2)} \end{bmatrix}$$

  - Computations so far require 5 multiplications

- Example-10 (cont'd)
  - Then

$$s(p) = \sum_{i=0}^{2} s^{(i)}(p) N^{(i)}(p) M^{(i)}(p) \bmod m^{(i)}(p)$$

$$= \left[s_0^{(0)} \left(\frac{p^3+p^2+p+1}{4} \right) + s_0^{(1)} \left(\frac{p^3-p^2+p-1}{-4} \right) + s_0^{(2)} \left(\frac{p^2-1}{-2} \right) \right] + s_1^{(2)} \left(p \cdot \frac{p^2-1}{-2} \right)$$

$$= \left(\frac{s_0^{(0)}}{4} + \frac{s_0^{(1)}}{4} + \frac{s_0^{(2)}}{2} \right) + p \left(\frac{s_0^{(0)}}{4} - \frac{s_0^{(1)}}{4} + \frac{s_1^{(2)}}{2} \right) + p^2 \left(\frac{s_0^{(0)}}{4} + \frac{s_0^{(1)}}{4} - \frac{s_0^{(2)}}{2} \right)$$

$$+ p^3 \left(\frac{1}{4} s_0^{(0)} - \frac{1}{4} s_0^{(1)} - \frac{1}{2} s_1^{(2)} \right)$$

  - So, we have

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{4} s_0^{(0)} \\ \frac{1}{4} s_0^{(1)} \\ \frac{1}{2} s_0^{(2)} \\ \frac{1}{2} s_1^{(2)} \end{bmatrix}$$

- Example-10 (cont'd)
  - Notice that:

$$\begin{bmatrix} \frac{1}{4}s_0^{(0)} \\ \frac{1}{4}s_0^{(1)} \\ \frac{1}{2}s_0^{(2)} \\ \frac{1}{2}s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \cdot$$

$$\begin{bmatrix} \frac{1}{4}h_0^{(0)} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4}h_0^{(1)} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}h_0^{(2)} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}\left(h_1^{(2)} - h_0^{(2)}\right) & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}\left(h_0^{(2)} - h_1^{(2)}\right) \end{bmatrix} \cdot \begin{bmatrix} x_0^{(0)} \\ x_0^{(1)} \\ x_0^{(2)} + x_1^{(2)} \\ x_0^{(2)} \\ x_1^{(2)} \end{bmatrix}$$

- Example-10 (cont'd)
  - Therefore, we have

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 & 0 \end{bmatrix} \cdot$$

$$\begin{bmatrix} \frac{h_0+h_1+h_2+h_3}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0-h_1+h_2-h_3}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0-h_2}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{-h_0+h_1+h_2-h_3}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1-h_2-h_3}{2} \end{bmatrix} \cdot$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

- Example-10 (cont'd)
  - This algorithm requires 5 multiplications and 15 additions
  - The direct implementation requires 16 multiplications and 12 additions (see the following matrix-form. Notice that the cyclic convolution matrix is a circulant matrix)

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

- An efficient cyclic convolution algorithm can often be easily extended to construct efficient linear convolution

- Example-11 (Example 8.5.2, p.249) Construct a 3X3 linear convolution using 4X4 cyclic convolution algorithm

- Example-11 (cont'd)
  - Let the 3-point coefficient sequence be $h=\{h_0,h_1,h_2\}$, and the 3-point data sequence be $x=\{x_0,x_1,x_2\}$
  - First extend them to 4-point sequences as:

$$h=\{h_0,h_1,h_2,0\},\quad x=\{x_0,x_1,x_2,0\}$$

  - Then the 3X3 linear convolution of h and x is

$$\Rightarrow h\cdot x=\begin{bmatrix} h_0x_0 \\ h_1x_0+h_0x_1 \\ h_2x_0+h_1x_1+h_0x_2 \\ h_2x_1+h_1x_2 \\ h_2x_2 \end{bmatrix} \qquad h\mathrm{O}_4x=\begin{bmatrix} h_0x_0+h_2x_2 \\ h_1x_0+h_0x_1 \\ h_2x_0+h_1x_1+h_0x_2 \\ h_2x_1+h_1x_2 \end{bmatrix}$$

  - The 4X4 cyclic convolution of h and x, i.e. $h\mathrm{O}_4x$, is:

- Example-11 (cont'd)
  - Therefore, we have $s(p) = h(p) \cdot x(p) = h\mathrm{O}_n x + h_2 x_2 (p^4 - 1)$
  - Using the result of Example-10 for $h\mathrm{O}_4 x$, the following convolution algorithm for 3X3 linear convolution is obtained:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 \\ 1 & -1 & 1 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

(continued on the next page)

- Example-11 (cont'd)

$$\cdot \begin{bmatrix} \frac{h_0+h_1+h_2}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0-h_1+h_2}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0-h_2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-h_0+h_1+h_2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1-h_2}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & h_2 \end{bmatrix}$$

$$\cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

- Example-11 (cont'd)
  - So, this algorithm requires 6 multiplications and 16 additions

- Comments:
  - In general, an efficient linear convolution can be used to obtain an efficient cyclic convolution algorithm. Conversely, an efficient cyclic convolution algorithm can be used to derive an efficient linear convolution algorithm

Design of fast convolution algorithm by inspection

- When the Cook-Toom or the Winograd algorithms can not generate an efficient algorithm, sometimes a clever factorization by inspection may generate a better algorithm
- Example-12 (Example 8.6.1, p.250) Construct a 3X3 fast convolution algorithm by inspection
  - The 3X3 linear convolution can be written as follows, which requires 9 multiplications and 4 additions

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} h_0 x_0 \\ h_1 x_0 + h_0 x_1 \\ h_2 x_0 + h_1 x_1 + h_0 x_2 \\ h_2 x_1 + h_1 x_2 \\ h_2 x_2 \end{bmatrix}$$

- Example-12 (cont'd)
  - Using the following identities:

$$s_1 = h_1 x_0 + h_0 \cdot x_1 = (h_0 + h_1) \cdot (x_0 + x_1) - h_0 x_0 - h_1 x_1$$
$$s_2 = h_2 x_0 + h_1 x_1 + h_0 x_2 = (h_0 + h_2)(x_0 + x_2) - h_0 x_0 + h_1 x_1 - h_2 x_2$$
$$s_3 = h_2 x_1 + h_1 x_2 = (h_1 + h_2)(x_1 + x_2) - h_1 x_1 - h_2 x_2$$

  - The 3X3 linear convolution can be written as:

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot$$

(continued on the next page)

- Example-12 (cont'd)

$$\cdot \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & h_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_0 + h_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 + h_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & h_1 + h_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

  - Conclusion: This algorithm, which can not be obtained by using the Cook-Toom or the Winograd algorithms, requires 6 multiplications and 10 additions