VLSI Signal Processing

Lecture 6 Fast Algorithms for Digital Signal Processing

Algorithm Strength Reduction

- Motivation
  - The number of strong operations, such as multiplications, is reduced possibly at the expense of an increase in the number of weaker operations, such as additions.
- Reduce computation complexity
- Example: Complex multiplication
  - (a+jb)(c+jd)=e+jf, a,b,c,d,e,f $\in$ **R**
  - The direct implementation requires 4 multiplications and 2 additions

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

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

$$ac - bd = a(c-d) + d(a-b)$$
$$ad + bc = b(c+d) + d(a-b)$$

3 multiplications
5 additions

Review of Digital Signal Processing

- Given two sequences:
  - Data sequence d_i, $0 \le i \le N-1$, of length N
  - Filter sequence g_i, $0 \le i \le L-1$, of length L
- Linear convolution

NL multiplications

$$s_i = \sum_{k=0}^{N-1} g_{i-k} d_k, \text{ or } s_i = \sum_{k=0}^{L-1} g_k d_{i-k}, \quad i = 0,1,\ldots,L+N-2$$

- Express the convolution in the notation of polynomials

$$d(x) = \sum_{i=0}^{N-1} d_i x^i, \quad g(x) = \sum_{i=0}^{L-1} g_i x^i. \text{ Then}$$

$$s(x) = g(x)d(x) = d(x)g(x), \text{ where } s(x) = \sum_{i=0}^{L+N-2} s_i x^i$$

CooK-Toom Algorithm

- An algorithm for linear convolution by using multiplying polynomials.
- Consider the following system

$$x \longrightarrow \boxed{h} \longrightarrow s$$

L-point sequence

N-point sequence

(L+N-1)-point sequence

given

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

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

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

To find $s_{L+N-2}, \ldots, s_1, s_0$ ➔ By solving L+N-1 linear equations

Review of Polynomial Ring

- For a field **F**, there is a polynomial ring **F**[x] called the ring of polynomials over **F**.
- Mathematical expression

$$f(x)=f_n x^n + f_{n-1}x^{n-1} + \dots + f_1 x + f_0,\ f_0,f_1,\dots,f_n \in \mathbf{F}$$

- If $f_n \neq 0$, then the degree of polynomial f(x) is n
- β is called a zero of polynomial f(x), if $\beta \in \mathbf{F}$ and $f(\beta)=0$.
- At most n field elements are zeros of a polynomial of degree n, otherwise, it is a zero polynomial
- Lagrange Interpolation
  - Let $\beta_0,\dots,\beta_n$ be a set of distinct elements, and let $p(\beta_k)$, k=0,…,n, be given. There is exactly one polynomial p(x) of degree n or less that has value $p(\beta_k)$, k=0, …, n. p(x) is given by

$$p(x)=\sum_{i=0}^{n} p(\beta_i)\frac{\prod_{j\neq i}(x-\beta_j)}{\prod_{j\neq i}(\beta_i-\beta_j)}$$

Cook-Toom (CT) Algorithm

1. Choose $L+N-1$ different real numbers $\beta_0, \beta_1, \cdots \beta_{L+N-2}$
2. Compute $h(\beta_i)$ and $x(\beta_i)$, for $i=\{0,1,\cdots,L+N-2\}$
3. Compute $s(\beta_i)=h(\beta_i)\cdot x(\beta_i)$, for $i=\{0,1,\cdots,L+N-2\}$
4. Compute $s(p)$ by using $s(p)=\sum_{i=0}^{L+N-2} s(\beta_i)\frac{\prod_{j\neq i}(p-\beta_j)}{\prod_{j\neq i}(\beta_i-\beta_j)}$

- 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

Example: 2 by 2 CT Algorithm

- 2 by 2 convolution in polynomial multiplication from is s(p)=h(p)x(p), where $h(p)=h_0+h_1p$, $x(p)=x_0+x_1p$, and $s(p)=s_0+s_1p+s_2p^2$
- Direct implementation:
  - require 4 multiplications and 1 addition

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

- CT algorithm
  - Step 1: Choose $\beta_0=0$, $\beta_1=1$, $\beta_2=-1$
  - Step 2 :

$$\begin{aligned} \beta_0 &= 0, & h(\beta_0) &= h_0, & x(\beta_0) &= x_0; \\ \beta_0 &= 1, & h(\beta_1) &= h_0 + h_1, & x(\beta_1) &= x_0 + x_1; \\ \beta_0 &= -1, & h(\beta_2) &= h_0 - h_1. & x(\beta_2) &= x_0 - x_1 \end{aligned}$$

Require no multiplications

2 by 2 CT Algorithm

- Step 3 : Calculate $s(\beta_0)$, $s(\beta_1)$, $s(\beta_2)$.

$$s(\beta_0)=h(\beta_0)x(\beta_0)$$
$$s(\beta_1)=h(\beta_1)x(\beta_1)$$
$$s(\beta_2)=h(\beta_2)x(\beta_2)$$

Require 3 multiplications

- Step 4: Compute s(p) by using Lagrange interpolation theorem

$$s(p)=s(\beta_0)\frac{(p-\beta_1)(p-\beta_2)}{(\beta_0-\beta_1)(\beta_0-\beta_2)}+s(\beta_1)\frac{(p-\beta_0)(p-\beta_1)}{(\beta_1-\beta_0)(\beta_1-\beta_2)}$$
$$+s(\beta 2)\frac{(p-\beta_0)(p-\beta_1)}{(\beta_2-\beta_0)(\beta_2-\beta_1)}$$
$$=s(\beta_0)+p(\frac{s(\beta_1)-s(\beta_2)}{2})+p^2(-s(\beta_0)+\frac{s(\beta_1)+s(\beta_2)}{2})$$
$$=s_0+ps_1+p^2s_2$$

2 by 2 CT Linear Convolution

– The preceding computation leads to the following matrix form

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} s(\beta_0) \\ s(\beta_1)/2 \\ s(\beta_2)/2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} h(\beta_0) & 0 & 0 \\ 0 & h(\beta_1)/2 & 0 \\ 0 & 0 & h(\beta_2)/2 \end{bmatrix} \cdot \begin{bmatrix} x(\beta_0) \\ x(\beta_1) \\ x(\beta_2) \end{bmatrix}$$

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

– 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. $s_0 = S_0, \quad s_1 = S_1 - S_2, \quad s_2 = -S_0 + S_1 + S_2$

Remarks

- Direct implementation needs 4 mutiplications and 1 addition
- If we take sequence h_i as filter coefficients and sequence x_i as the signal sequence, then the terms H_i need not be recomputed each time the filter is used. They can be precomputed once off-line and stored.
- 2 by 2 CT algorithm needs 3 multiplications and 5 additions (ignoring the additions in the pre-computation).
- The number of multiplications is reduced by 1 at the expense of 4 extra additions

Remarks

- Some additions in the preaddition or postaddition matrix can be shared. When we count the number of additions, we only count one instead of two or three.
- As can be seen from examples, the CT algorithm can be understood as a matrix decomposition

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad \text{or} \quad s = T \cdot x$$

C H D

Cook-Toom Algorithm

- Generally, the equation can be expresses as s=**T**x=**CHD**x
  - **C** is called the postaddition matrix and **D** the preaddition matrix. **H** is a diagonal matrix with H_i, i=0, 1, …, L+N-2 on the diagonal
- Since **T**=**CHD**, it implies that the CT algorithm provides a way to factorize the convolution matrix **T** into three multiplying matrices and the total number of multiplications is determined by the non-zero elements on the main diagonal of the matrix **H** (note matrices **C** and **D** contain only small integers)
- 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

Concluding Remarks

- The Cook-Toom algorithm is efficient as measured by the number of multiplications
- As the size of the problem increases, the number of additions increase rapidly
- The choices of β_i=0, $\pm$1 are good, while the choices of $\pm$2, $\pm$4 (or other small integers) result in complicated pre-addition and post-addition matrices.
- For larger problems, CT algorithm becomes cumbersome
- ➔Winograd Algorithm

Review of Integer Ring (1)

- For every integer c and positive integer d, there is a unique pair of integer Q, called the quotient, and integer s, the remainder, such that $c=dQ+s$, where $0 \le s \le d-1$
- Notation: $Q=\lfloor c/d \rfloor$, $s=R_d[c]$
- Euclidean Algorithm: Given two positive integers s and t, t<s, their GCD can be computed by an iterative application of the division algorithm.

$$
\begin{aligned}
s &= Q^{(1)}t + t^{(1)} \\
t &= Q^{(2)}t^{(1)} + t^{(2)} \\
t^{(1)} &= Q^{(3)}t^{(2)} + t^{(3)} \\
&\vdots \\
t^{(n-2)} &= Q^{(n)}t^{(n-1)} + t^{(n)} \\
t^{(n-1)} &= Q^{(n-1)}t^{(n)}
\end{aligned}
$$

1. the process stops when a remainder of zero is obtained.
2. The last nonzero remainder $t^{(n)}$ is the GCD(s,t)
3. Matrix notation expression

$$
\begin{bmatrix} s^{(r)} \\ t^{(r)} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)} \end{bmatrix} \begin{bmatrix} s^{(r-1)} \\ t^{(r-1)} \end{bmatrix}
$$

Review of Integer Ring (2)

- For any integer s and t, there exists integers a and b s.t. GCD[s,t]= as + bt
- It is possible to uniquely determine a nonnegative integer given its moduli with respect to each of several integers, provided that the integer is known to be smaller than the product of the moduli.
- Example: $\{m_1=3, m_2=5\}$ $M=3\times5=15$

	→	M_1	M_2
0	→	0	0
1	→	1	1
2	→	2	2
3	→	0	3
4	→	1	4

	→	M_1	M_2
5	→	2	0
6	→	0	1
7	→	1	2
8	→	2	3
9	→	0	4

	→	M_1	M_2
10	→	1	0
11	→	2	1
12	→	0	2
13	→	1	3
14	→	2	4

Unique representation

Example

$\{m_1, m_2, m_3\} = \{3,4,5\}$ $\boldsymbol{M} = 3 \times 4 \times 5 = 60$

	$\rightarrow$	M_1	M_2	M_3
0	$\rightarrow$	0	0	0
1	$\rightarrow$	1	1	1
2	$\rightarrow$	2	2	2
3	$\rightarrow$	0	3	3
4	$\rightarrow$	1	0	4
5	$\rightarrow$	2	1	0
6	$\rightarrow$	0	2	1
7	$\rightarrow$	1	3	2
8	$\rightarrow$	2	0	3
9	$\rightarrow$	0	1	4
10	$\rightarrow$	1	2	0
11	$\rightarrow$	2	3	1

	$\rightarrow$	M_1	M_2	M_3
12	$\rightarrow$	0	0	2
13	$\rightarrow$	1	1	3
14	$\rightarrow$	2	2	4
15	$\rightarrow$	0	3	0
16	$\rightarrow$	1	0	1
17	$\rightarrow$	2	1	2
18	$\rightarrow$	0	2	3
19	$\rightarrow$	1	3	4
20	$\rightarrow$	2	0	0
21	$\rightarrow$	0	1	1
22	$\rightarrow$	1	2	2
23	$\rightarrow$	2	3	3

	$\rightarrow$	M_1	M_2	M_3
24	$\rightarrow$	0	0	4
25	$\rightarrow$	1	1	0
26	$\rightarrow$	2	2	1
27	$\rightarrow$	0	3	2
28	$\rightarrow$	1	0	3
29	$\rightarrow$	2	1	4
30	$\rightarrow$	0	2	0
31	$\rightarrow$	1	3	1
32	$\rightarrow$	2	0	2
33	$\rightarrow$	0	1	3
34	$\rightarrow$	1	2	4
35	$\rightarrow$	2	3	0

	$\rightarrow$	M_1	M_2	M_3
36	$\rightarrow$	0	0	1
37	$\rightarrow$	1	1	2
38	$\rightarrow$	2	2	3
39	$\rightarrow$	0	3	4
40	$\rightarrow$	1	0	0
41	$\rightarrow$	2	1	1
42	$\rightarrow$	0	2	2
43	$\rightarrow$	1	3	3
44	$\rightarrow$	2	0	4
45	$\rightarrow$	0	1	0
46	$\rightarrow$	1	2	1
47	$\rightarrow$	2	3	2

	$\rightarrow$	M_1	M_2	M_3
48	$\rightarrow$	0	0	3
49	$\rightarrow$	1	1	4
50	$\rightarrow$	2	2	0
51	$\rightarrow$	0	3	1
52	$\rightarrow$	1	0	2
53	$\rightarrow$	2	1	3
54	$\rightarrow$	0	2	4
55	$\rightarrow$	1	3	0
56	$\rightarrow$	2	0	1
57	$\rightarrow$	0	1	2
58	$\rightarrow$	1	2	3
59	$\rightarrow$	2	3	4

Chinese Remainder Theorem (1)

- Given a set of integers m_0, m_1, …, m_k that are pair-wise relatively prime (co-prime), then for each integer c, $0 \leq c < M = m_0 m_{1 \ldots} m_k$, there is a one-to-one map between c and the vector of residues

$$\left(R_{m_0}[c], R_{m_1}[c], \ldots, R_{m_k}[c]\right)$$

- Conversely, given a set of co-prime integers m_0, m_1, …, m_k and a set of integers c_0, c_1, …, c_k with $c_i < m_i$. Then the system of equations

$$c_i = c \ (\text{mod } m_i), \quad i=0,1,\ldots,k$$

has at most one solution for $0 \leq c < M$

Chinese Remainder Theorem (2)

- Define $M_i = M/m_i$, then $GCD[M_i, m_i]=1$. So there exists integers N_i and n_i with

  $$GCD[M_i, m_i] = 1 = N_i M_i + n_i m_i, \quad i=0,1,\ldots,k$$

- The system of equations $c_i = c \pmod{m_i}$, $0 \le i \le k$, is uniquely solved by

We need to find N_i

$$c = \sum_{i=0}^{k} \underline{c_i}\, N_i \underline{M_i} \pmod{M}$$

given

GCD Example

- GCD(993,186)

993	186
930	126
63	60
60	60
3	0

$993 = 5 \times 186 + 63$

$186 = 2 \times 63 + 60$

$63 = 1 \times 60 + 3$

$60 = 20 \times 3 + 0$

$$
\begin{aligned}
\underline{GCD(993,186)} &= 3 \\
&= 63 - 1 \times 60 \\
&= 63 - 1 \times (186 - 2 \times 63) \\
&= 3 \times 63 - 1 \times 186 \\
&= 3 \times (993 - 5 \times 186) - 1 \times 186 \\
&= \underline{3 \times 993 - 16 \times 186}
\end{aligned}
$$

Remark

1. $\boxed{N_i M_i + n_i m_i = GCD(M_i, m_i)} = 1$, then we have

$$N_i M_i = 1 \quad (\bmod m_i)$$

2. $$c \ (\bmod m_i) = \sum_{i=0}^{k} c_i N_i M_i \ (\bmod m_i)$$
$$= c_i N_i M_i \ (\bmod m_i)$$
$$= c_i \ (\bmod m_i)$$

Example

- m_0=3, m_1=4, m_2=5. Then by Euclidean theorem, we have

N_i

$$
\begin{aligned}
m_0 &= 3, \quad M_0 = 20, \quad (-1)20 + (7)3 = 1; \\
m_1 &= 4, \quad M_1 = 15, \quad (-1)15 + (4)4 = 1; \\
m_2 &= 5, \quad M_0 = 12, \quad (-2)12 + (5)5 = 1;
\end{aligned}
$$

- The integer c can be calculated as

n_i

$$
\begin{aligned}
c &= \sum_{i=0}^{k} c_i N_i M_i \pmod{M} \\
&= \left(-20c_0 - 15c_1 - 24c_2\right) \pmod{60}
\end{aligned}
$$

- Example

$$c = 17, \text{i.e.} \left(c_0, c_1, c_2\right) = (2,1,2)$$

$$\text{Conversely, } c = (-20\times 2 - 15\times 1 - 24\times 2) \pmod{60} = 17$$

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Remarks

- By taking residues, large integers are broken down into small pieces (that may be easy to add and multiply)
- Examples:

$$\begin{array}{rllll} 7\rightarrow(1, & 3, & 2 &) \\ +3\rightarrow(0, & 3, & 3 &) \\ \hline 10\rightarrow(1 \bmod 3, & 6 \bmod 4, & 5 \bmod 5) = (1,2,0) \end{array}$$

$$\begin{array}{rllll} 7\rightarrow(1, & 3, & 2 &) \\ \times 3\rightarrow(0, & 3, & 3 &) \\ \hline 21\rightarrow(0 \bmod 3, & 9 \bmod 4, & 6 \bmod 5) = (0,1,1) \end{array}$$

CRT for Polynomials (1)

- Given a set of polynomials $m^{(0)}(x)$, $m^{(1)}(x)$, …, $m^{(k)}(x)$, that are pair-wise relatively prime (co-prime), then for each polynomial $c(x)$, $\deg(c(x))<\deg(M(x))$, $M(x)= m^{(0)}(x)m^{(1)}(x)\ldots m^{(k)}(x)$, there is a one-to-one map between $c(x)$ and the vector of residues

$$(R_{m^{(0)}(x)}[c(x)], R_{m^{(1)}(x)}[c(x)], \ldots, R_{m^{(k)}(x)}[c(x)])$$

- Conversely, given a set of co-prime polynomials $m^{(0)}(x)$, $m^{(1)}(x)$, …, $m^{(k)}(x)$ and a set of polynomials $c^{(0)}(x)$, $c^{(1)}(x)$, …, $c^{(k)}(x)$ with $\deg(c^{(i)}(x))< \deg(m^{(i)}(x))$. Then the system of equations

$$c^{(i)}(x) = c(x) \pmod{m^{(i)}(x)}, \quad i=0,1,\ldots,k$$

has at most one solution for $\deg(c(x)) < \deg(M(x))$

Chinese Remainder Theorem (2)

- Define $M^{(i)}(x)=M(x)/\ m^{(i)}(x)$, then $GCD[M^{(i)}(x),m^{(i)}(x)]=1$. So there exists polynomials $N^{(i)}(x)$ and $n^{(i)}(x)$ with
  $GCD[M^{(i)}(x),m^{(i)}(x)]=1= N^{(i)}(x)M^{(i)}(x)+n^{(i)}(x)m^{(i)}(x)$, $i=0,1,\ldots,k$
- The system of equations $c^{(i)}(x) =c(x) \pmod{m^{(i)}(x)}$, $0\le i\le k$, is uniquely solved by

$$c(x)=\sum_{i=0}^{k} c^{(i)}(x)\,N^{(i)}(x)\,M^{(i)}(x) \pmod{M(x)}$$

Remarks

- The remainder of a polynomial with regard to modulus $x^i + f(x)$, where $\deg(f(x)) < i$, can be evaluated by substituting x^i by $-f(x)$ in the polynomial
- Example

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

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

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

Winograd Algorithm

- Recall that we wish to compute $s(p)=h(p)x(p)$ for linear convolution
- Consider the following system:

$$s(p)=h(p)x(p) \mod m(p)$$

- As long as $\deg(s)<\deg(m)$, then the system can be used for solving linear convolution problem
- If $m(p)= m^{(0)}(p)m^{(1)}(p)\ldots m^{(k)}(p)$. Efficient implementation for linear convolution can be constructed using the CRT by choosing and factoring the polynomial $m(p)$ appropriately.

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: $s^{(i)}(p) = h^{(i)}(p)x^{(i)}(p) \bmod m^{(i)}(p), \qquad for \quad i = 0,1,\cdots,k$

  This step requires multiplications

- 5. Compute $s(p)$ by using:

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

Example: 2×3 Winograd Algorithm

- The linear convolution h(p)x(p) has degree 3

  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	$p-1$	$p^3 + p$	$-\frac{1}{2}(p^2+p+2)$	$\frac{1}{2}$
2	p^2+1	$p^2 - p$	$-\frac{1}{2}(p-2)$	$\frac{1}{2}(p-1)$

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

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

Require
no multiplication

Example: 2×3 Winograd Algorithm

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

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

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

- 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:

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

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

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

Example: 2×3 Winograd Algorithm

– 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: 2×3 Winograd Algorithm

– 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:

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

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

Remarks

- It requires 5 multiplications and 11 additions, compared with 6 multiplications and 2 additions.
- In above example, the order in which the additions are done is unspecified.
- One can experiment with the order of the additions to minimize their number, however, there is no theory developed to aid in doing this.
- The number of multiplications is highly dependent on the degree of m(p)
- The degree of m(p) should be as small as possible. By CRT, the extreme case is deg(m(p))=deg(s(p))+1
- Let $s'(p)=s(p)- h_{N-1}x_{L-1}\, m(p)$. Note that s'(p) (mod m(p)) = s(p) (mod m(p))
- Modified Winograd Algorithm:
  - Choose m(p) with a degree equal to that of s(p)
  - Apply CRT to s'(p)
  - $s(p)=s'(p)+ h_{N-1}x_{L-1}\, m(p)$.

Iterated Convolution

- To make use of efficient short-length convolution algorithms iteratively, one can build long convolutions
- These algorithms do not achieve minimal multiplication complexity, but achieve a good balance between multiplications and addition complexity
- Iterated Convolution algorithm
  - Decompose the long convolution algorithm for short convolutions
  - Construct fast convolution algorithm for short convolutions
  - Use the short convolution algorithms to iteratively (or hierarchically) implement the long convolution

Example

- 4×4 linear convolution algorithm
  - 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 $\quad h'_0(p) = h_0 + h_1 p, \quad h'_1(p) = h_2 + h_3 p$

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

$$\Rightarrow \begin{aligned} h(p) &= \underline{h'_0(p)} + \underline{h'_1(p)} p^2, \quad i.e., h(p) = h(p,q) = h'_0(p) + h'_1(p)q \\ x(p) &= x'_0(p) + x'_1(p)p^2, \quad i.e., x(p) = x(p,q) = x'_0(p) + x'_1(p)q \end{aligned}$$

$$\begin{aligned} s(p) = h(p)x(p) &= h(p,q)x(p,q) \\ \Rightarrow \quad &= [h'_0(p) + h'_1(p)q] \cdot [x'_0(p) + x'_1(p)q] \\ &= h'_0(p)x'_0(p) + [h'_0(p)x'_1(p) + h'_1(p)x'_0(p)]q + h'_1(p)x'_1(p)q^2 \\ &= s'_0(p) + s'_1(p)q + s'_2(p)q^2 = s(p,q) \end{aligned}$$

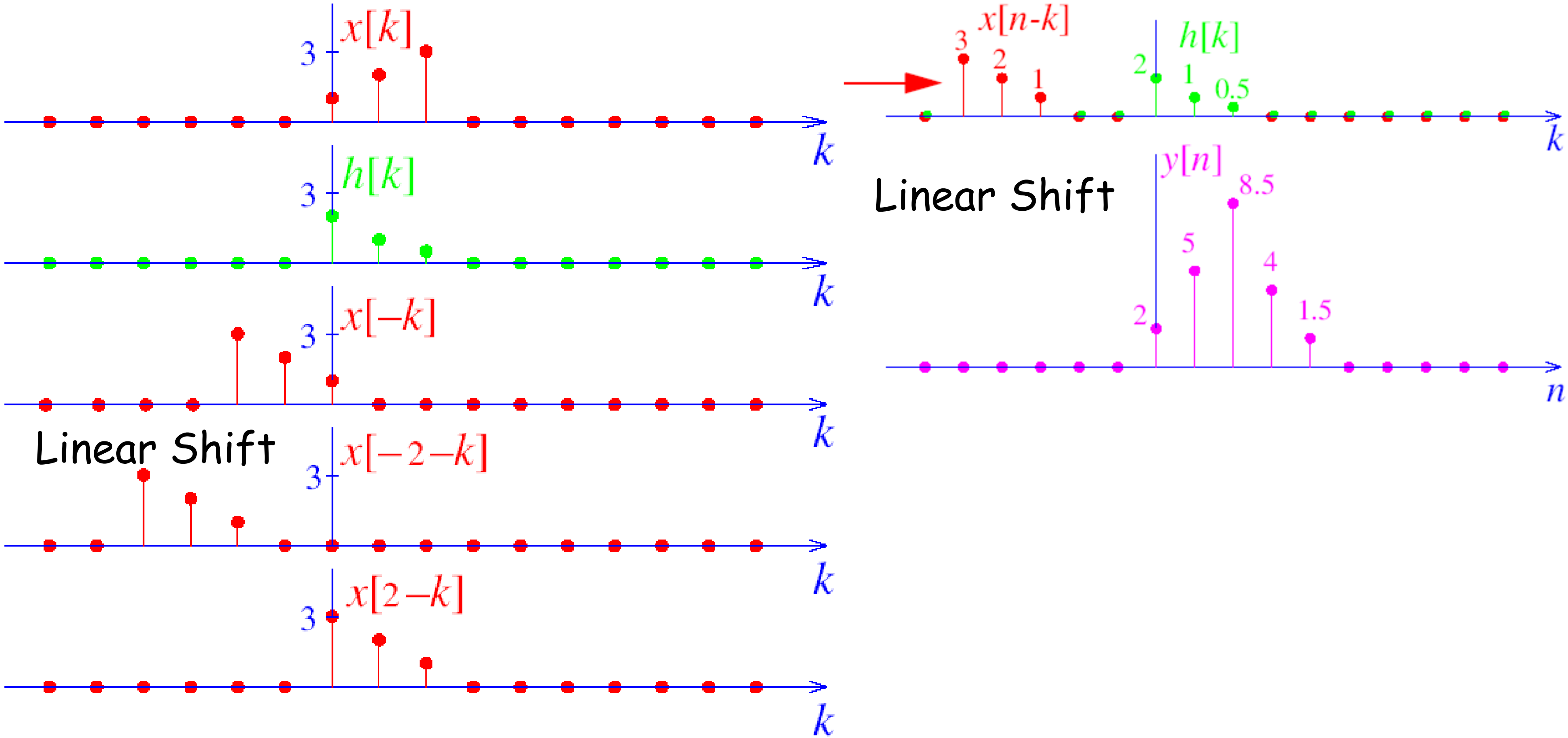
Remarks

- The 4×4 convolution is decomposed into two levels of nested 2×2 short convolutions
- The top-level, which is expressed in terms of variable q, can be using by 2×2 convolution algorithms

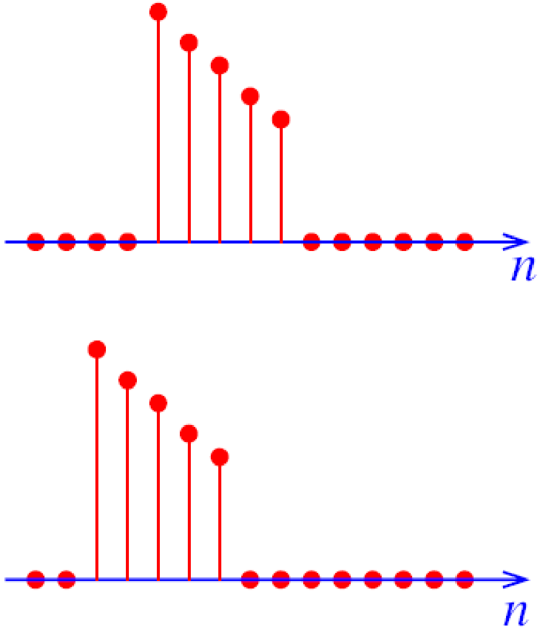
$$\begin{bmatrix} s_0'(p) \\ s_1'(p) \\ s_2'(p) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_0'(p) & 0 & 0 \\ 0 & h_0'(p)-h_1'(p) & 0 \\ 0 & 0 & h_1'(p) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0'(p) \\ x_1'(p) \end{bmatrix}$$

- The polynomial multiplications, for computing s'_0, s'_1, s'_2, are again 2×2 convolutions, i.e. the second level 2×2 short convolutions

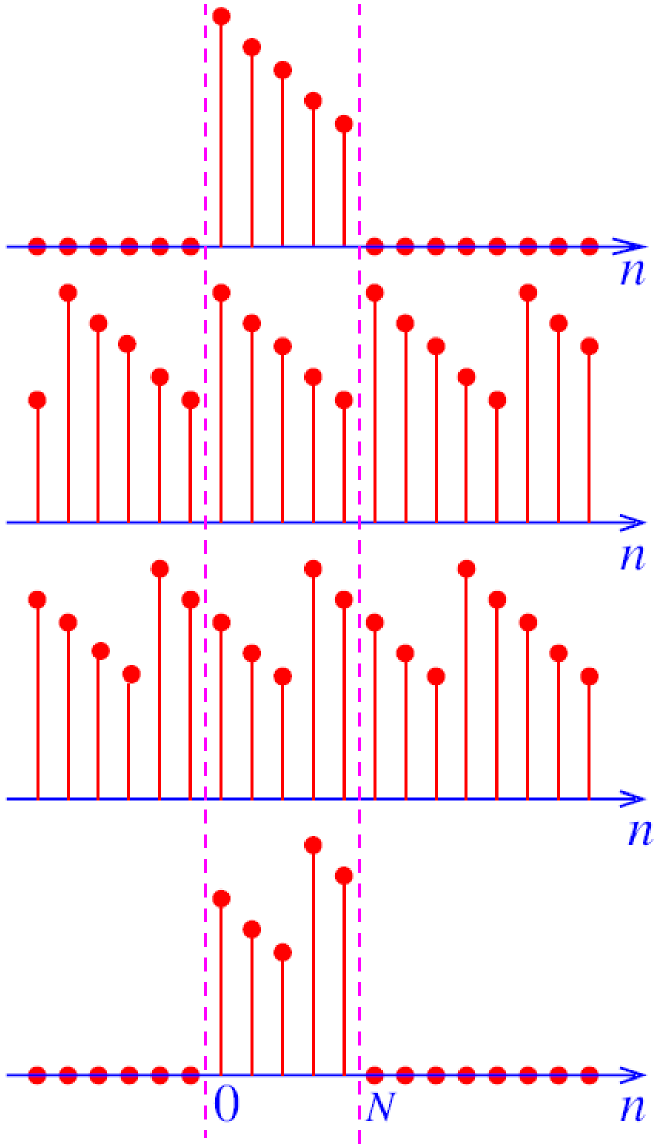
Linear Convolution



Circular Shift



Conventional shift
(linear shift)



Circular Convolution

- Given two sequences x_i and h_i, $0 \le i \le n-1$, of block length n
- Notation: $((n-k)) \equiv n-k \pmod{n}$
- Cyclic (or circular) convolution s'_i, $0 \le i \le n-1$, is given by

$$s'_i = \sum_{k=0}^{n-1} h_{((i-k))} x_k$$

Coefficients with indices larger than n-1 are folded back into terms with indices small than n

We can express the cyclic convolution by polynomial product :

$$s'(p) = s(p) \ \left(\text{mod } p^n - 1\right) = h(p)x(p) \ \left(\text{mod } p^n - 1\right)$$

Direct Implementation

- Consider the following system

Circular Convolution

$$x \longrightarrow \boxed{O_h} \longrightarrow s$$

n-point sequence (x)

n-point sequence (O_h)

n-point sequence (s)

4×4 cyclic convolution

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

Circular shift

16 multiplications
12 additions

Remarks

- The cyclic convolution can be computed as a linear convolution reduced by modulo p^n-1
- There are $2n-1$ outputs of linear convolution, while there are n outputs of cyclic convolution
- The cyclic convolution can be computed by using CRT with $m(p)=p^n-1$

Example: 4×4 Cyclic Convolution

- 4×4 cyclic convolution by using m(p)=p^4-1
  - 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

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

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

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

: multiplication

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

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

– Since

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

– 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

- Then

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

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

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

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

- 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

– Notice that:

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

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

– Therefore, we have

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

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

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

5 multiplications
15 additions

Discrete Fourier Transform

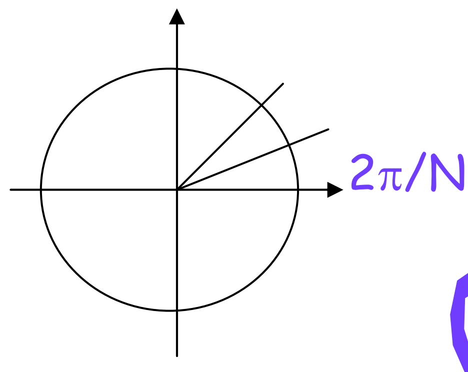
- Discrete Fourier transform (DFT) pairs

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0,1,\ldots,N-1$$

N complex multiplications
N-1 complex additions

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad n = 0,1,\ldots,N-1,$$

$$\text{where } W_N^{-kn} = e^{-j\frac{2\pi}{N}kn}$$



- DFT/IDFT can be implemented by using the same hardware
- It requires N^2 complex multiplications and N(N-1) complex additions

Decimation in Time

$$X_N[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{kn} \qquad \longleftarrow \quad N\text{–point DFT}$$

$$= \sum_{\boxed{n \text{ even}}} x[n]\, W_N^{kn} + \sum_{\boxed{n \text{ odd}}} x[n]\, W_N^{kn}$$

n → 2ℓ, 2ℓ+1

$$= \sum_{l=0}^{N/2-1} x[2l]\, W_N^{2lk} + \sum_{l=0}^{N/2-1} x[2l+1]\, W_N^{(2l+1)k}$$

$$= \sum_{l=0}^{N/2-1} x[2l][\underbrace{W_N^2}_{W_{N/2}}]^{lk} + \underline{W_N^k} \sum_{l=0}^{N/2-1} x[2l+1][\underbrace{W_N^2}_{W_{N/2}}]^{lk}$$

twiddle factor

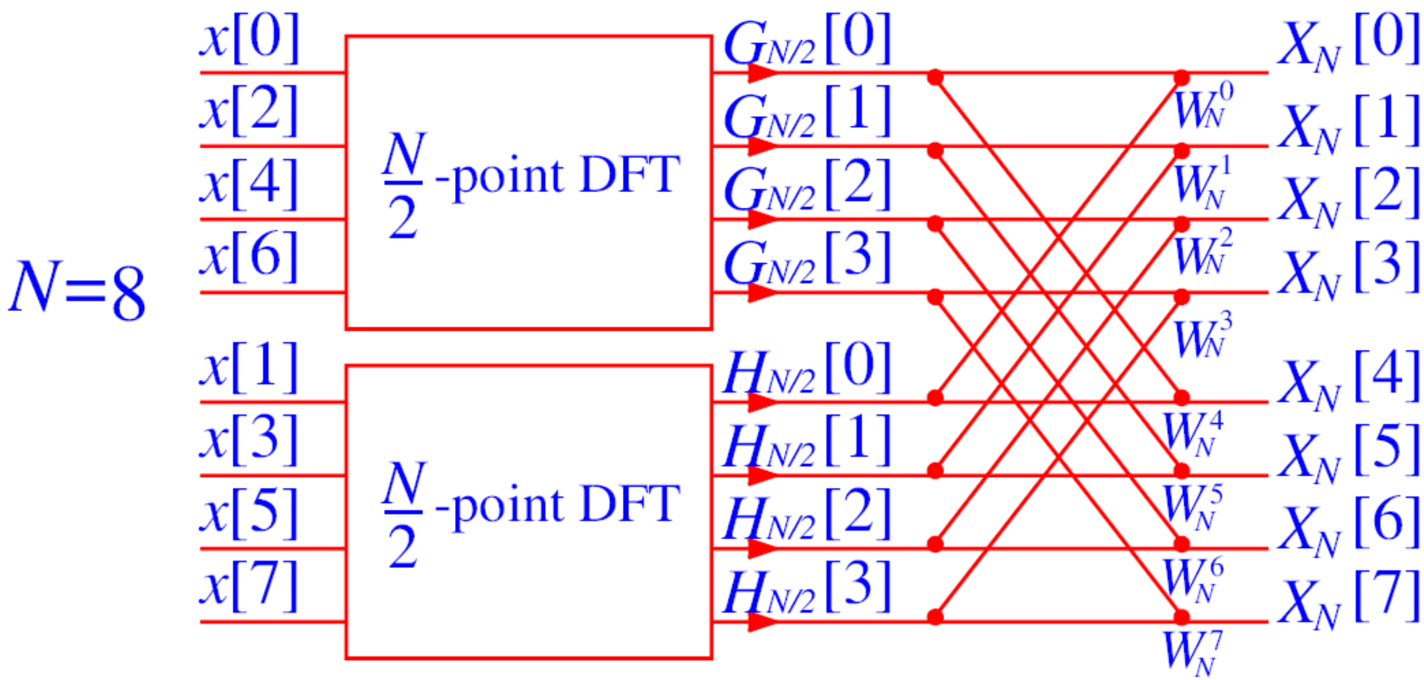
$$= \underbrace{\sum_{l=0}^{N/2-1} x[2l]\, W_{N/2}^{lk} + W_N^k \sum_{l=0}^{N/2-1} x[2l+1]\, W_{N/2}^{lk}}_{\text{two } N/2\text{–point DFT's!!!}}$$

$N+2(N/2)^2$ complex multiplications vs. N^2 complex multiplication

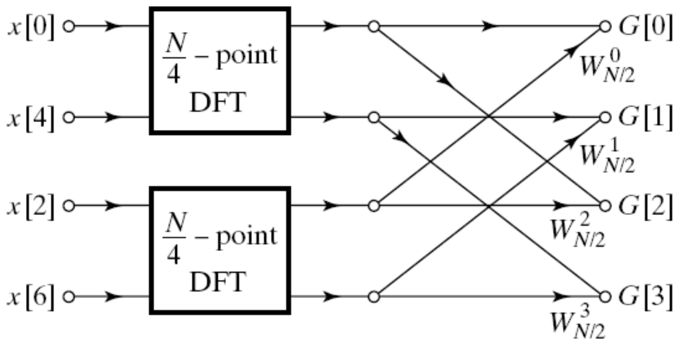
Using a briefer system of notation:

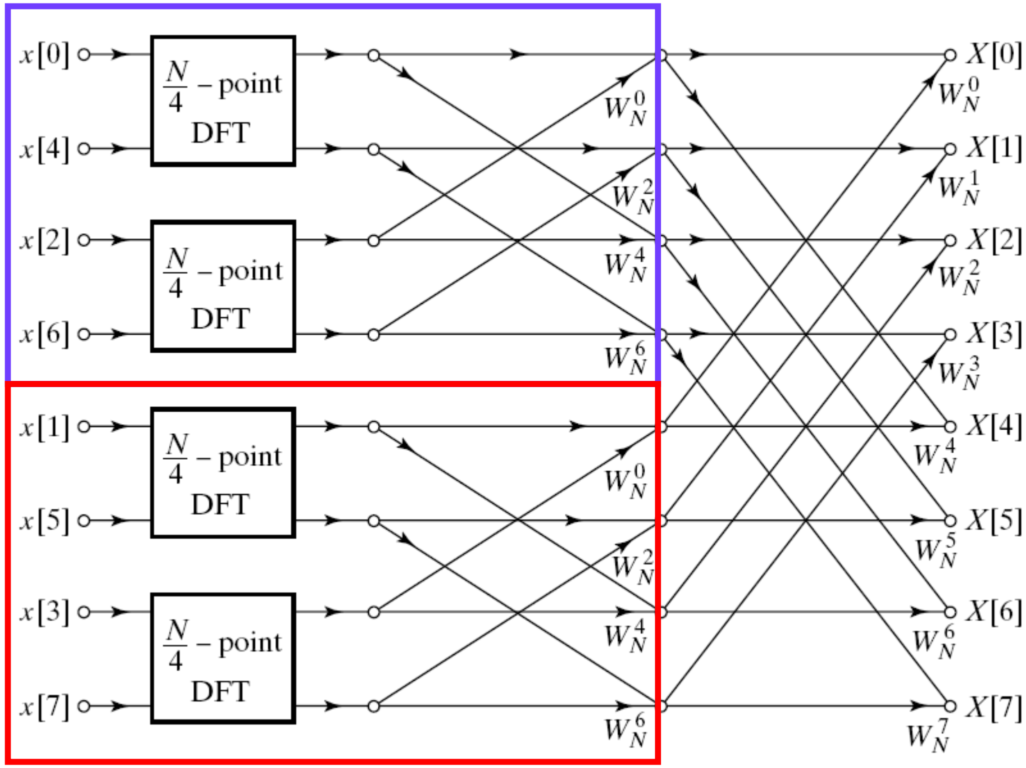
$$\boxed{X_N[k] = G_{N/2}[k] + W_N^k H_{N/2}[k]\,,}$$

where $G_{N/2}[k]$ and $H_{N/2}[k]$ are the $N/2$-point DFTs involving $x[n]$ with even and odd n, respectively.



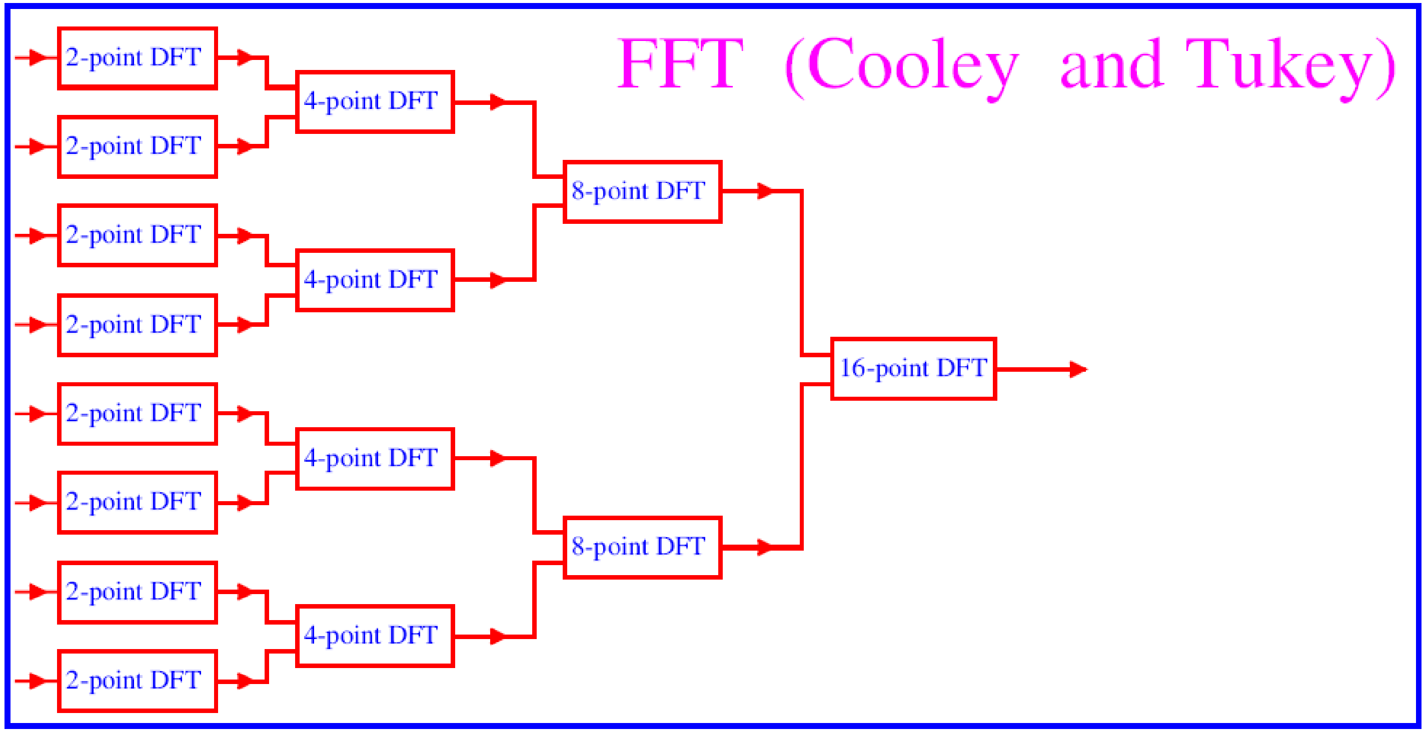
Flow Graph of the DIT FFT



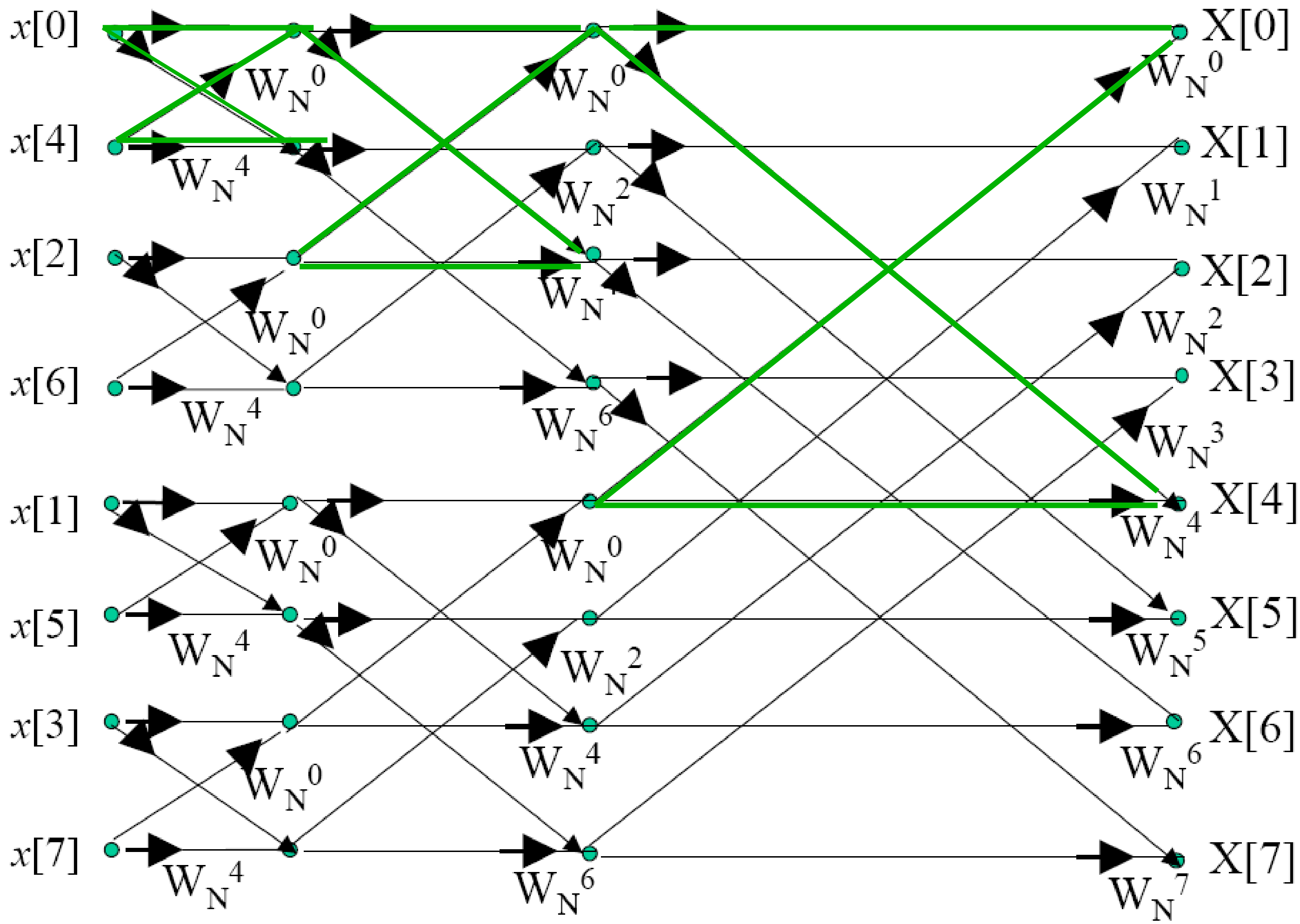


Corollary:

Any N-point DFT with even N can be computed via two $N/2$-point DFTs. In turn, if $N/2$ is even then each of these $N/2$-point DFTs can be computed via two $N/4$-point DFTs and so on. In the case of $N = 2^r$, all N, $N/2$, $N/4$ … are even and such a process of "splitting" ends up with all 2-point DFTs!



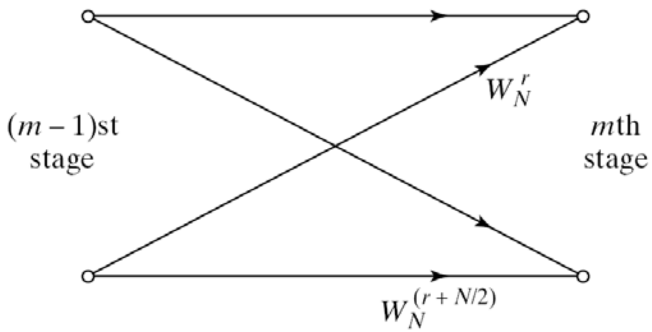
8-point DIT DFT



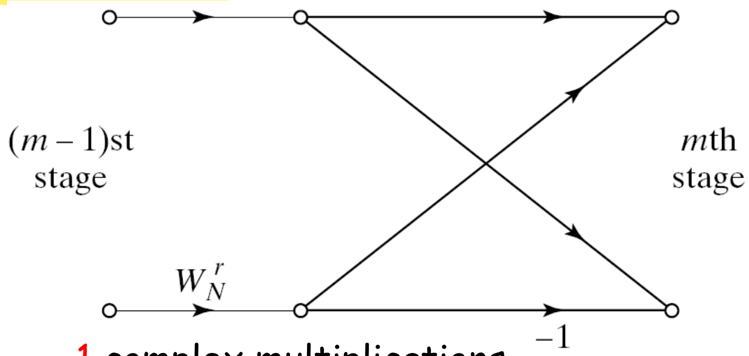
Remarks

- It requires $v=\log_2 N$ stages. Each stage has N/2 butterfly operation (radix-2 DIT FFT), which requires 2 complex multiplications and 2 complex additions
- Each stage has N complex multiplications and N complex additions
- The number of complex multiplications (as well as additions) is equal to $N \log_2 N$
- By symmetry property, we have (butterfly operation)

$$W_N^{r+N/2} = W_N^r W_N^{N/2} = W_N^r e^{-j\pi} = -W_N^{N/2}$$

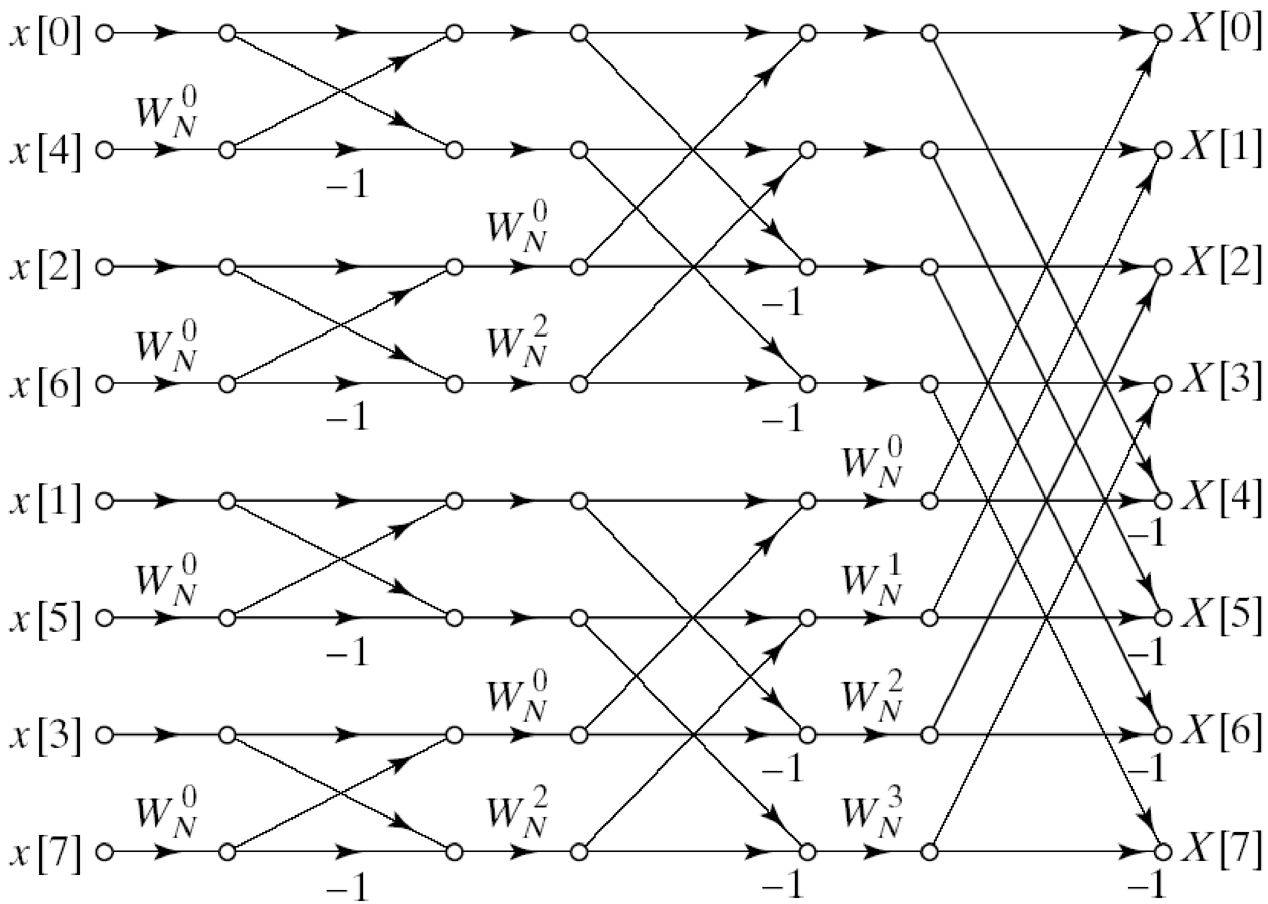


2 complex multiplications
2 complex additions



1 complex multiplications
2 complex additions

8-point FFT

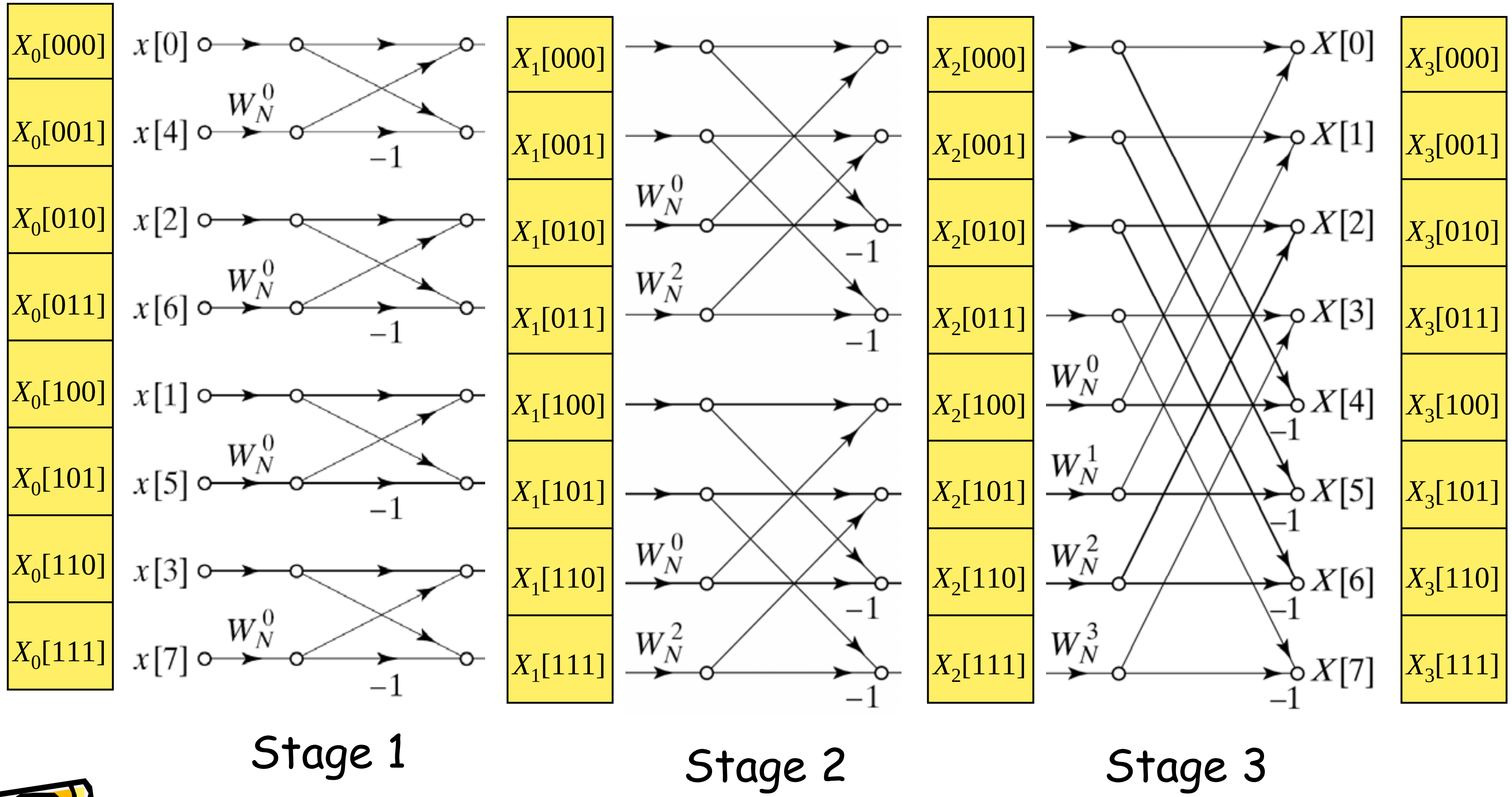


Bit-Reversed order

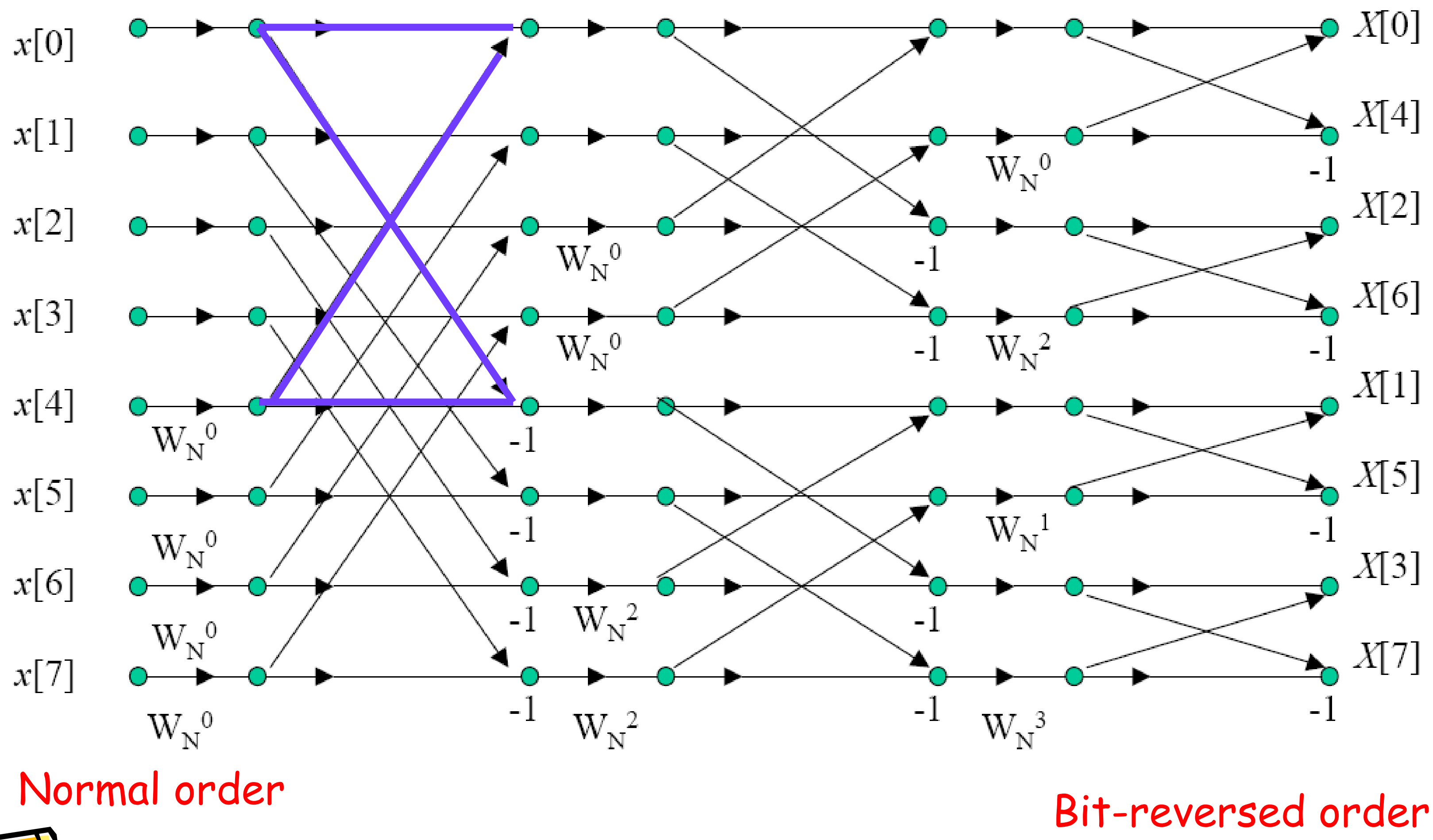
Normal order

In-Place Computation

The same register array can be used in each stage

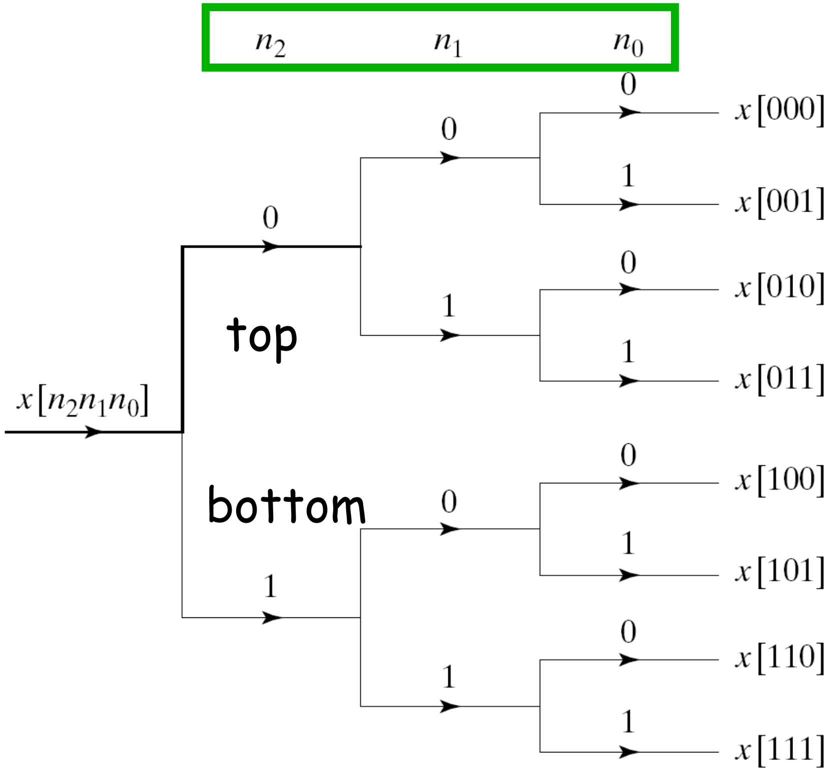


8-point FFT

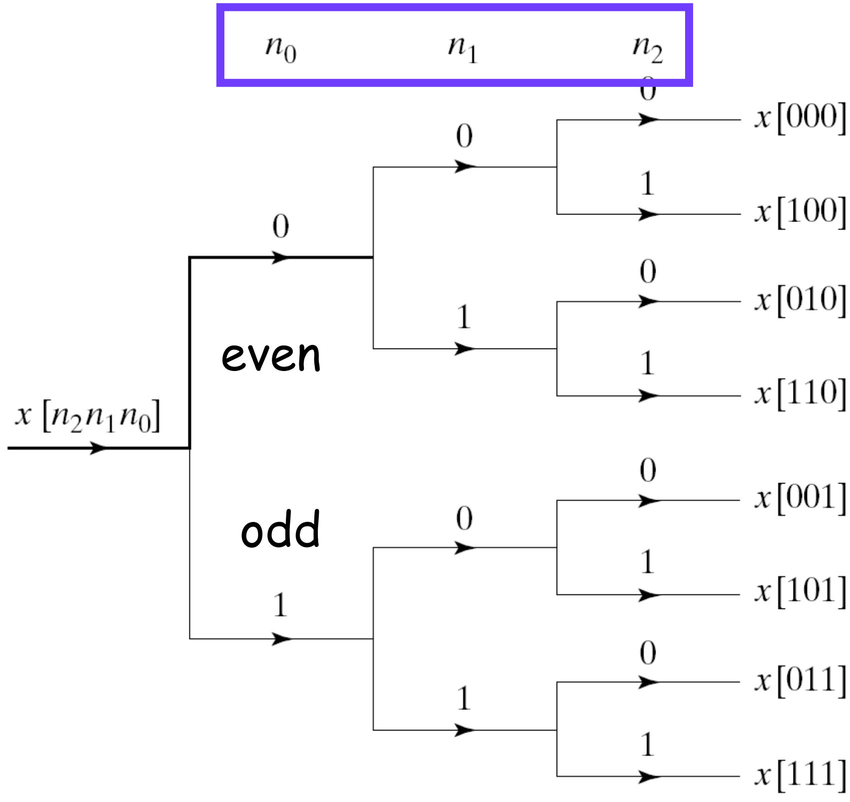


x[0]
x[1]
x[2]
x[3]
x[4]
x[5]
x[6]
x[7]
W_N^0
W_N^0
W_N^0
W_N^0
-1
-1
-1
-1
W_N^0
W_N^0
W_N^2
W_N^2
-1
-1
-1
-1
W_N^0
W_N^2
W_N^1
W_N^3
-1
-1
-1
-1
X[0]
X[4]
X[2]
X[6]
X[1]
X[5]
X[3]
X[7]
Normal order
Bit-reversed order

Normal-Order Sorting v.s. Bit-Reversed Sorting



Normal Order



Bit-reversed Order

DFT v.s. Radix-2 FFT

- DFT: N^2 complex multiplications and $N(N-1)$ complex additions
- Recall that each butterfly operation requires one complex multiplication and two complex additions
- FFT: $(N/2)\log_2 N$ multiplications and $N \log_2 N$ complex additions
- In-place computations: the input and the output nodes for each butterfly operation are horizontally adjacent ➔ only one storage arrays will be required

Decimation in Frequency (DIF)

- Recall that the DFT is $X[k]=\sum_{n=0}^{N-1}x[n]W_N^{nk},\ 0\le k\le N-1$
- DIT FFT algorithm is based on the decomposition of the DFT computations by forming small subsequences in time domain index "n": n=2ℓ or n=2ℓ+1
- One can consider dividing the output sequence X[k], in frequency domain, into smaller subsequences: k=2r or k=2r+1:

$$X[k]\left\langle \begin{array}{l} X[2r] \\ X[2r+1] \end{array}\right. \qquad 0\le r\le \tfrac{N}{2}-1$$

Substitution of variables

$$X[2r]=\sum_{n=0}^{\frac{N}{2}-1}x[n]W_N^{2nr}+\sum_{n=\frac{N}{2}}^{N-1}x[n]W_N^{2nr}=\sum_{n=0}^{N/2-1}x[n]W_N^{2nr}+\sum_{n=0}^{N/2-1}x[n+\tfrac{N}{2}]W_N^{2r(n+\frac{N}{2})}$$

$$=\sum_{n=0}^{N/2-1}(x[n]+x[n+\tfrac{N}{2}])W_{\frac{N}{2}}^{nr}$$

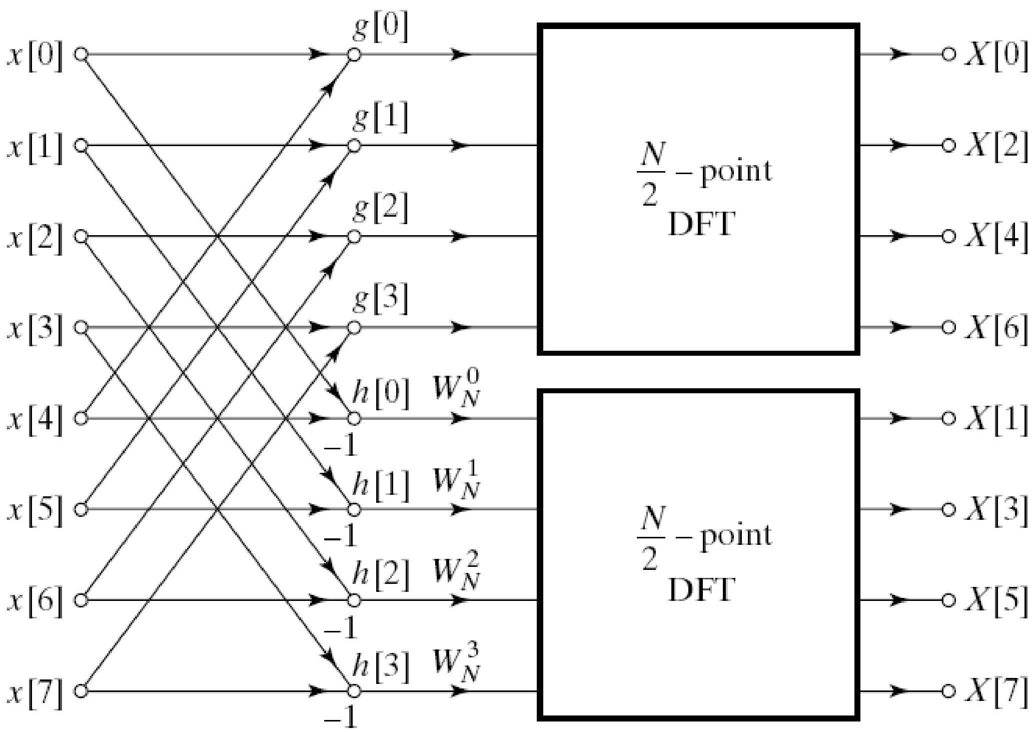
$$\boxed{W_N^{2r(n+\frac{N}{2})}=W_N^{2rn}W_N^{rN}=W_N^{2rn}}$$

DIF FFT Algorithm (1)

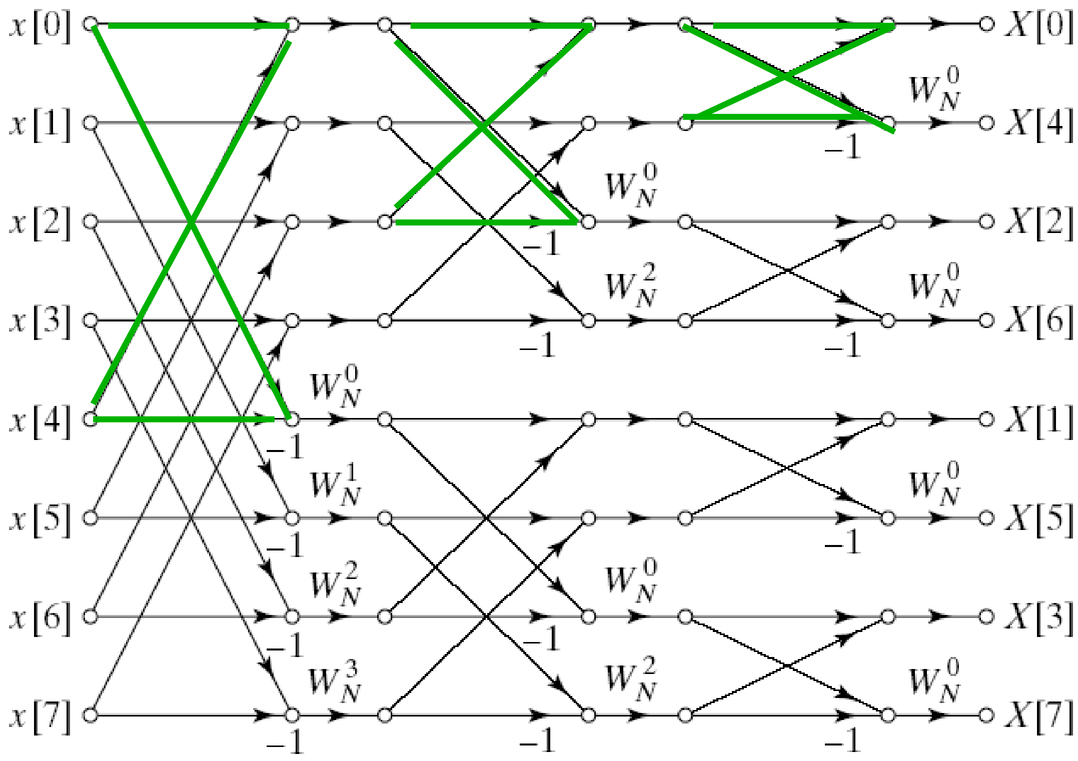
$$\boxed{0 \le r \le \tfrac{N}{2} - 1}$$

$$X[2r] = \sum_{n=0}^{N/2-1} \underline{(x[n] + x[n+\tfrac{N}{2}])} W_{\frac{N}{2}}^{nr}$$ is just N/2-point DFT. Similarly,

$$X[2r+1] = \sum_{n=0}^{N/2-1} (x[n] - x[n+\tfrac{N}{2}])\, W_N^{n(2r+1)} = \sum_{n=0}^{N/2-1} \underline{\{x[n] - x[n+\tfrac{N}{2}]\}\, W_N^n}\, W_{N/2}^{nr}$$



DIF FFT Algorithm (2)

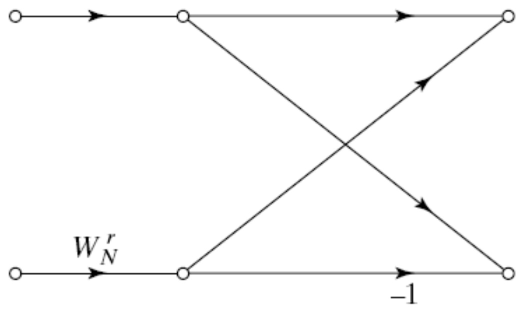


v=$\log_2$N stages, each stage has N/2 butterfly operation.

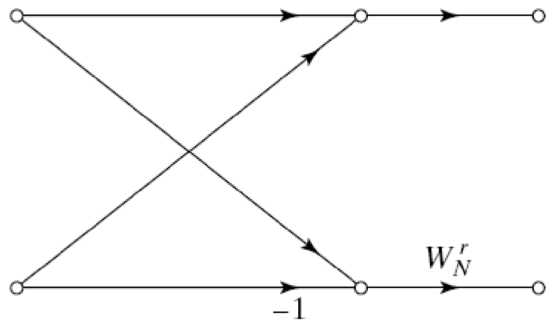
(N/2)$\log_2$N complex multiplications, N complex additions

Remarks

- The basic butterfly operations for DIT FFT and DIF FFT respectively are transposed-form pair.



DIT BF unit



DIF BF unit

- The I/O values of DIT FFT and DIF FFT are the same
- Applying the transpose transform to each DIT FFT algorithm, one obtains DIF FFT algorithm

Fast Convolution with the FFT

- Given two sequences x_1 and x_2 of length N_1 and N_2 respectively
  - Direct implementation requires N_1N_2 complex multiplications
- Consider using FFT to convolve two sequences:
  - Pick N, a power of 2, such that $N \geq N_1+N_2-1$
  - Zero-pad x_1 and x_2 to length N
  - Compute N-point FFTs of zero-padded x_1 and x_2, then we obtain X_1 and X_2
  - Multiply X_1 and X_2
  - Apply the IFFT to obtain the convolution sum of x_1 and x_2
  - Computation complexity: $2(N/2)\log_2 N + N + (N/2)\log_2 N$

Implementation Issues

- Radix-2, Radix-4, Radix-8, Split-Radix,Radix-2^2, …,
- I/O Indexing
- In-place computation
  - Bit-reversed sorting is necessary
  - Efficient use of memory
  - Random access (not sequential) of memory. An address generator unit is required.
  - Good for cascade form: FFT followed by IFFT (or vice versa)
    - E.g. fast convolution algorithm
- Twiddle factors
  - Look up table
  - CORDIC rotator

Algorithm Strength Reduction

- Motivation
  - The number of strong operations, such as multiplications, is reduced possibly at the expense of an increase in the number of weaker operations, such as additions.
- Reduce computation complexity
- Example: Complex multiplication
  - (a+jb)(c+jd)=e+jf, a,b,c,d,e,f $\in$ **R**
  - The direct implementation requires 4 multiplications and 2 additions

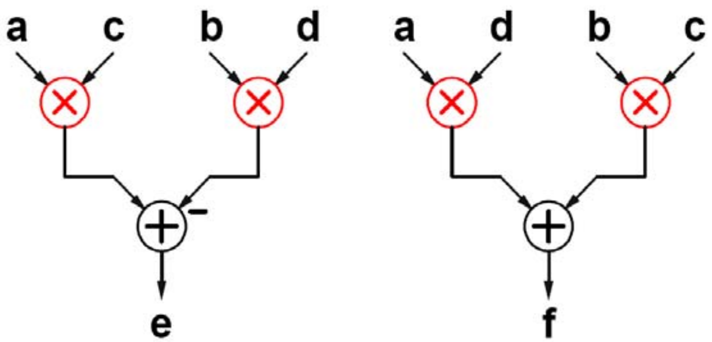
$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

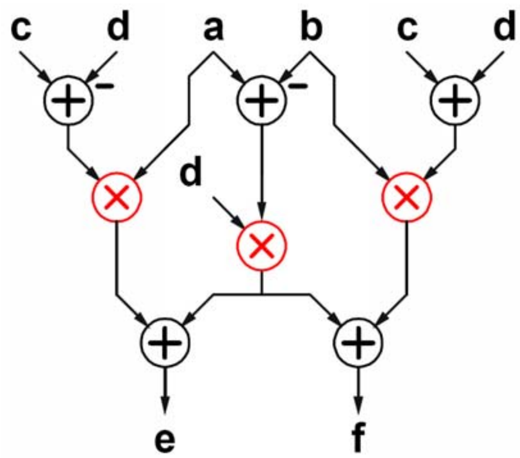
  - However, the number of multiplication can be reduced to 3 at the expense of 3 extra additions by using the identities

$$ac - bd = a(c-d) + d(a-b)$$
$$ad + bc = b(c+d) + d(a-b)$$

3 multiplications
5 additions

Complex Multiplication





Reduce the number of strong operation (less switched capacitance), however, increase the critical path

➔ Speed?, Area?, Power? ....

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

FIR Filters

$$x(1), x(2), x(3), \ldots \longrightarrow \boxed{H(z)} \longrightarrow$$

$$y(n) = h(n) * x(n) \Leftrightarrow Y(z) = H(z)X(z)$$

Time-domain

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 \\ h_1 & h_0 & 0 \\ 0 & h_1 & h_0 \\ 0 & 0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Transform-domain

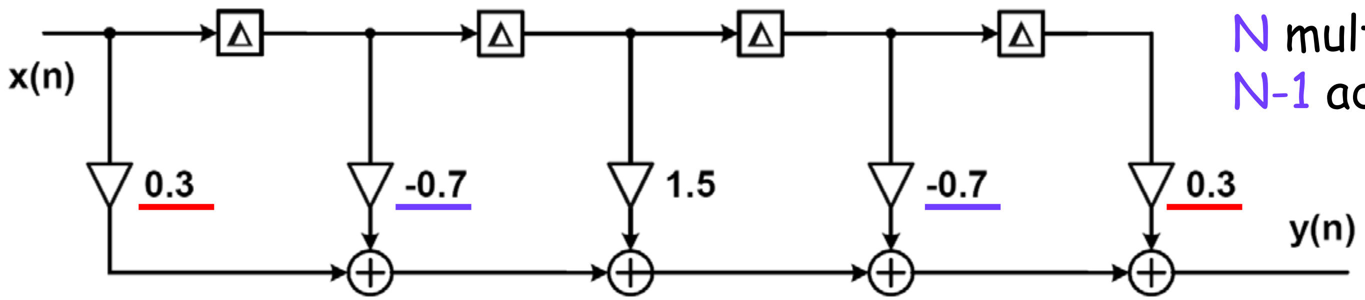
$$Y(z) = H(z) \cdot X(z) = \left(\sum_{n=0}^{N-1} h(n) z^{-n} \right) \cdot \left(\sum_{n=0}^{\infty} x(n) z^{-n} \right)$$

$$h(n) * x(n) = \sum_{i=0}^{N-1} h(i) x(n-i), \quad n = 0,1,2,\cdots,\infty$$

Example: Linear Phase FIR

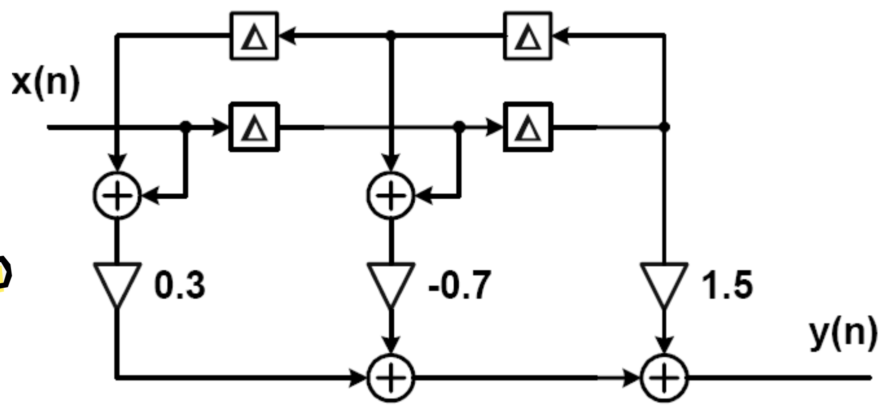
Linear phase FIR filter: with approximately constant frequency-response magnitude and linear phase (constant group delay) in pass-band

N-tap



N multipliers
N-1 adders

By exploiting substructure sharing to reduce area



(N+1)/2 multipliers
N-1 adders, if odd N

N/2 multipliers
N-1 adders, if even N

An Efficient Decomposition

- Example: 2-fold decomposition

$$H(z) = h[0] + h[1]z^{-1} + h[2]z^{-2} + h[3]z^{-3} + h[4]z^{-4} + h[5]z^{-5} + h[6]z^{-6}$$
$$= \underbrace{(h[0] + h[2]z^{-2} + h[4]z^{-4} + h[6]z^{-6})}_{H_0(z^2)} + z^{-1}\underbrace{(h[1] + h[3]z^{-2} + h[5]z^{-4})}_{H_1(z^2)}$$

- Example 3-fold decomposition

$$H(z) = h[0] + h[1]z^{-1} + h[2]z^{-2} + h[3]z^{-3} + h[4]z^{-4} + h[5]z^{-5} + h[6]z^{-6}$$
$$= \underbrace{(h[0] + h[3]z^{-3} + h[6]z^{-6})}_{H_0(z^3)} + z^{-1}\underbrace{(h[1] + h[4]z^{-3})}_{H_1(z^3)} + z^{-2}\underbrace{(h[2] + h[5]z^{-3})}_{H_2(z^3)}$$

- General case (N-fold decomposition)

$$H(z) = \sum_{k=-\infty}^{\infty} h[k]z^{-k} = \sum_{l=0}^{N-1} z^{-l} H_l(z^N), \text{ where } H_l(z) = \sum_{k=-\infty}^{\infty} h[Nk+l]z^{-k}$$

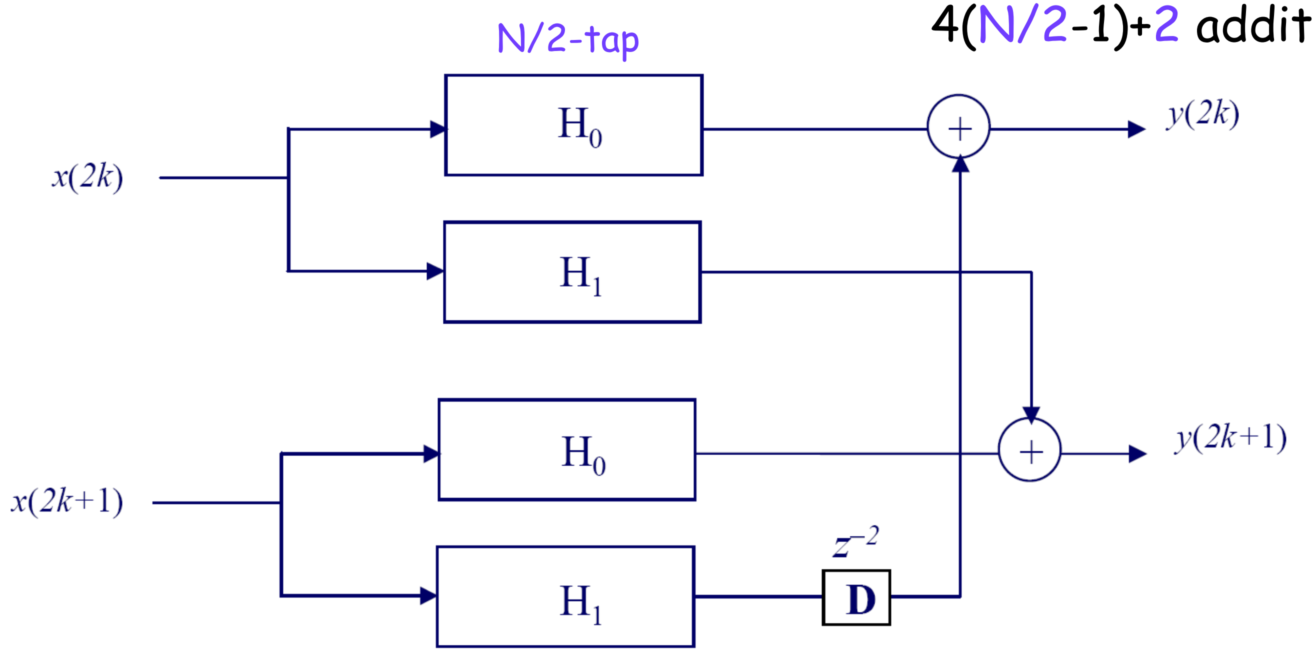
Traditional Parallel Architecture

- 2-fold parallel architecture

$$X(z) = X_0(z^2) + z^{-1}X_1(z^2) \qquad H(z) = H_0(z^2) + z^{-1}H_1(z^2)$$

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

4(N/2) multiplications
4(N/2-1)+2 additions



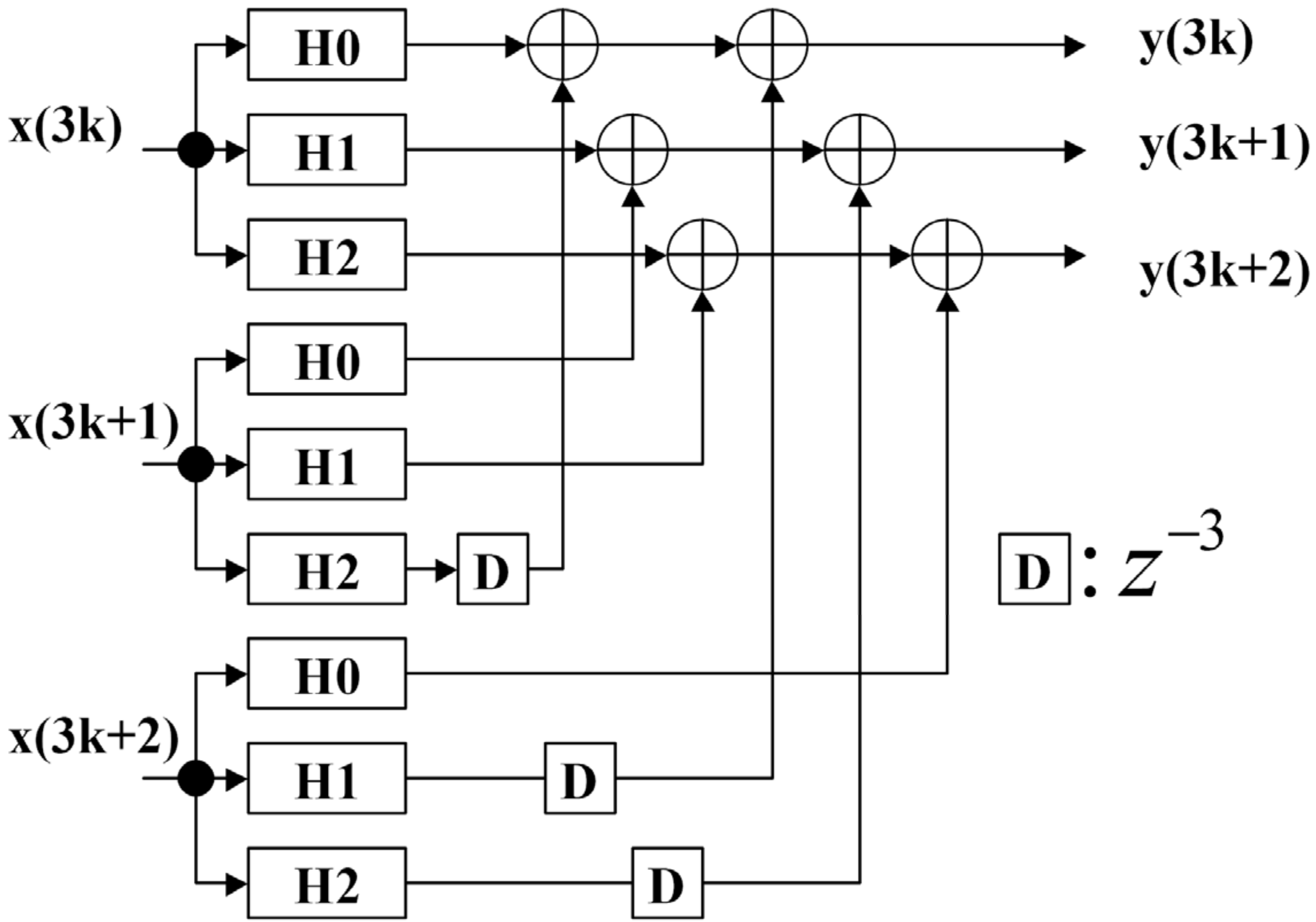
Traditional Parallel FIR

$$\begin{bmatrix} Y_0 \\ Y_1 \\ \cdots \\ Y_{L-1} \end{bmatrix} = \begin{bmatrix} H_0 & z^{-L}H_{L-1} & \cdots & z^{-L}H_1 \\ H_1 & H_0 & \cdots & z^{-L}H_2 \\ \cdots & \cdots & \cdots & \cdots \\ H_{L-1} & H_{L-2} & \cdots & H_0 \end{bmatrix} \cdot \begin{bmatrix} X_0 \\ X_1 \\ \cdots \\ X_{L-1} \end{bmatrix}$$

L-parallel FIR filter of length N/L requires

1. L^2 (N/L) multiplications, i.e. LN multiplications
2. L^2 (N/L -1) +L(L-1) additions, i.e. L(N-1) additions

~ LN multiply-add operations


H0
H1
H2
H0
H1
H2
D
H0
H1
D
H2
D
x(3k)
x(3k+1)
x(3k+2)
y(3k)
y(3k+1)
y(3k+2)
D : z^{-3}

Fast FIR Algorithm (FFA)

- First by applying L-fold polyphase decomposition for H(z)
  - There are L filters of length N/L
- By applying Winograd algorithm
  - 2 polynomials of degree L-1 can be implemented by using 2L-1 product terms.
  - Each product terms are equivalent to filtering operations in the block formulation
  - Consequently, it can be realized using approximately (2L-1) FIR filters of length N/L

➔ It requires 2N-N/L multiplications