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**
  - \((a+jb)(c+jd)=e+jf, \ a,b,c,d,e,f \in \mathbb{R}\)
  - The direct implementation requires 4 multiplications and 2 additions

  \[
  \begin{bmatrix}
  c \\
  d
  \end{bmatrix}
  =
  \begin{bmatrix}
  e \\
  f
  \end{bmatrix}
  \begin{bmatrix}
  a \\
  b
  \end{bmatrix}
  \]

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

  \[
  ac - bd = a(c - d) + d(a - b) \quad \text{3 multiplications}
  \]

  \[
  ad + bc = b(c + d) + d(a - b) \quad \text{5 additions}
  \]
Review of Digital Signal Processing

- Given two sequences:
  - Data sequence \( d_i, 0 \leq i \leq N-1 \), of length \( N \)
  - Filter sequence \( g_i, 0 \leq i \leq L-1 \), of length \( L \)

- Linear convolution

\[
\sum_{k=0}^{L-1} g_{i-k}d_k, \quad s_i = \sum_{k=0}^{L-1} g_kd_{i-k}, \quad i = 0,1,\ldots,L+N-2
\]

- Express the convolution in the notation of polynomials

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

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

CooK-Toom Algorithm

- An algorithm for linear convolution by using multiplying polynomials.

- Consider the following system

\[
x \xrightarrow{h} s
\]

\[
\begin{align*}
x(p) &= h_{N-1}p^{N-1} + \ldots + h_1p + h_0 \\
\end{align*}
\]

\[
\begin{align*}
\text{given } &\quad h(p) = h_{N-1}p^{N-1} + \ldots + h_1p + h_0 \\
\text{To find } &\quad s_{L+N-2}, \ldots, s_1, s_0 \Rightarrow \text{By solving } L+N-1 \text{ linear equations}
\end{align*}
\]
Review of Polynomial Ring

- For a field \( F \), there is a polynomial ring \( F[x] \) called the ring of polynomials over \( F \).
- Mathematical expression
  \[ f(x) = f_n x^n + f_{n-1} x^{n-1} + \ldots + f_1 x + f_0, \quad f_0, f_1, \ldots, f_n \in F \]
- If \( f_n \neq 0 \), then the degree of polynomial \( f(x) \) is \( n \)
- \( \beta \) is called a zero of polynomial \( f(x) \), if \( \beta \in F \) 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 \( \beta_0, \beta_1, \ldots, \beta_n \) be a set of distinct elements, and let \( p(\beta_k), \) \( k=0, \ldots, n \), be given. There is exactly one polynomial \( p(x) \) of degree \( n \) or less that has value \( p(\beta_k), \) \( k=0, \ldots, n \). \( p(x) \) is given by
    \[
    p(x) = \sum_{i=0}^{n} p(\beta_i) \prod_{j \neq i} \frac{(x - \beta_j)}{(\beta_i - \beta_j)}
    \]

Cook-Toom (CT) Algorithm

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

- Algorithm Complexity
  - The goal of the fast-convolution algorithm is to reduce the multiplication complexity. So, if \( \beta_i \)'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
  \( s(p) = h(p) \times x(p) \), where
  \( h(p) = h_0 + h_1 p, \ x(p) = x_0 + x_1 p, \) and
  \( s(p) = s_0 + s_1 p + s_2 p^2 \)

- Direct implementation:
  - require 4 multiplications and 1 addition

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

- CT algorithm
  - Step 1: Choose \( \beta_0 = 0, \beta_1 = 1, \beta_2 = -1 \)
  - Step 2: \[
  \begin{bmatrix}
  \beta_0 \\
  \beta_1 \\
  \beta_2
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

- CT algorithm
  - Step 3: Calculate \( s(\beta_0), s(\beta_1), s(\beta_2) \).
    \[
    \begin{align*}
    s(\beta_0) &= h(\beta_0) x(\beta_0) \\
    s(\beta_1) &= h(\beta_1) x(\beta_1) \\
    s(\beta_2) &= h(\beta_2) x(\beta_2)
    \end{align*}
    \]

- Step 4: Compute \( s(p) \) by using Lagrange interpolation theorem
  \[
  s(p) = s(\beta_0) \frac{(p - \beta_1)(p - \beta_2)}{(\beta_0 - \beta_1)(\beta_0 - \beta_2)}
  + s(\beta_1) \frac{(p - \beta_0)(p - \beta_2)}{(\beta_1 - \beta_0)(\beta_1 - \beta_2)}
  + s(\beta_2) \frac{(p - \beta_0)(p - \beta_1)}{(\beta_2 - \beta_0)(\beta_2 - \beta_1)}
  \]
  \[
  = s(\beta_0) + p \frac{(s(\beta_1) - s(\beta_2))}{2}
  + p^2 \left( -s(\beta_0) + s(\beta_1) + s(\beta_2) \right)
  = s_0 + p s_1 + p^2 s_2
  \]
2 by 2 CT Linear Convolution

The preceding computation leads to the following matrix form
\[
\begin{bmatrix}
  x_0 \\
  s_1 \\
  x_2
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & -1 \\
  -1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  x(\beta_2) \\
  x(\beta_1)/2 \\
  x(\beta_2)/2
\end{bmatrix}
\]

Remarks

- Direct implementation needs 4 multiplications 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:

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

\[s = T \cdot x\]

Cook-Toom Algorithm

- Generally, the equation can be expresses as \( s = T x = CHDx \)
  - \( C \) 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 $\beta_i=0, \pm 1$ 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.
- $\Rightarrow$ 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 $0 \leq s < d$.
- 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.

\[
\begin{align*}
S &= Q(1)t + r(1) \\
T &= Q(2)t + r(2) \\
T(r) &= Q(3)t + r(3) \\
&\vdots \\
T(r-2) &= Q^n t(r-1) + r^{(r)} \\
T(r-1) &= Q^n t^{(r-1)}
\end{align*}
\]

1. the process stops when a remainder of zero is obtained.
2. The last nonzero remainder $t^{(r)}$ 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. \( \text{GCD}(s,t) = as + bt \)
- 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: \( \{m_1=3,m_2=5\} \quad M=3\times5=15 \)

\[
\begin{array}{c|cccc|c|cccc|c}
0 & M_1 & M_2 & 0 & M_1 & M_2 & 0 & M_1 & M_2 \\
1 & 1 & 1 & 6 & 0 & 1 & 11 & 1 & 2 \\
2 & 2 & 2 & 7 & 0 & 1 & 12 & 0 & 2 \\
3 & 0 & 3 & 8 & 2 & 3 & 13 & 1 & 3 \\
4 & 1 & 4 & 9 & 0 & 4 & 14 & 2 & 4 \\
\end{array}
\]

Unique representation

Example

\[
\{m_1, m_1', m_1''\} = \{3,4,5\} \quad M = 3 \times 4 \times 5 = 60
\]

\[
\begin{array}{c|cccc|c|cccc|c}
0 & M_1 & M_1' & M_1'' & 0 & M_1 & M_1' & M_1'' & 0 & M_1 & M_1' & M_1'' \\
1 & 1 & 1 & 13 & 1 & 1 & 10 & 37 & 1 & 1 & 12 & 49 \\
2 & 2 & 2 & 14 & 2 & 2 & 24 & 2 & 2 & 21 & 38 & 50 \\
3 & 0 & 3 & 15 & 0 & 3 & 0 & 27 & 0 & 3 & 2 & 39 & 51 \\
4 & 1 & 0 & 16 & 1 & 0 & 1 & 28 & 1 & 0 & 3 & 40 & 52 \\
5 & 2 & 1 & 17 & 2 & 1 & 2 & 29 & 2 & 1 & 4 & 41 & 53 \\
6 & 0 & 2 & 18 & 0 & 2 & 3 & 30 & 0 & 2 & 0 & 42 & 54 \\
7 & 1 & 3 & 19 & 1 & 3 & 4 & 37 & 1 & 3 & 1 & 43 & 55 \\
8 & 2 & 0 & 20 & 2 & 0 & 0 & 32 & 2 & 0 & 2 & 44 & 56 \\
9 & 0 & 1 & 21 & 0 & 1 & 1 & 33 & 0 & 1 & 3 & 45 & 57 \\
10 & 1 & 2 & 22 & 1 & 2 & 2 & 34 & 1 & 2 & 4 & 46 & 58 \\
11 & 2 & 3 & 23 & 2 & 3 & 3 & 35 & 2 & 3 & 0 & 47 & 59 \\
\end{array}
\]
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 \( c, 0 \leq c < M = m_0m_1...m_k \), there is a one-to-one map between \( c \) and the vector of residues \( [R_{m_0}[c], R_{m_1}[c], ..., R_{m_k}[c]] \)

- 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
  \[ c_i = c \pmod{m_i}, \ i = 0, 1, ..., k \]
  has at most one solution for \( 0 \leq c < M \)

Chinese Remainder Theorem (2)

- Define \( M_i = M/m_i \), then \( \text{GCD}(M_i, m_i) = 1 \). So there exist integers \( N_i \) and \( n_i \) with
  \[ \text{GCD}(M_i, m_i) = 1 = N_iM_i + n_im_i, \ i = 0, 1, ..., k \]
- The system of equations \( c_i = c \pmod{m_i}, 0 \leq i \leq k \), is uniquely solved by
  \[ c = \sum_{i=0}^{k} c_iN_iM_i \pmod{M} \]
  We need to find \( N_i \)
  given
GCD Example

• $GCD(993, 186)$

\[
\begin{array}{c|c|c}
993 & 186 & 993 = 5 \times 186 + 63 \\
930 & 126 & 186 = 2 \times 63 + 60 \\
63 & 60 & 63 = 1 \times 60 + 3 \\
60 & 60 & 60 = 20 \times 3 + 0 \\
3 & 0 &
\end{array}
\]

\[GCD(993, 186) = 3\]

\[= 63 - 1 \times 60\]
\[= 63 - 1 \times (186 - 2 \times 63)\]
\[= 3 \times 63 - 1 \times 186\]
\[= 3 \times (993 - 5 \times 186) - 1 \times 186\]
\[= 3 \times 993 - 16 \times 186\]

Remark

1. \[N_i M_i + n_i m_i = GCD(M_i, m_i) = 1, \text{ then we have}\]
\[N_i M_i = 1 \pmod{m_i}\]

2. \[c \pmod{m_i} = \sum_{i=0}^{k} c_i N_i M_i \pmod{m_i}\]
\[= c_i N_i M_i \pmod{m_i}\]
\[= c_i \pmod{m_i}\]
**Example**

- \( m_0 = 3, m_1 = 4, m_2 = 5 \). Then by Euclidean theorem, we have
  \[
  m_0 = 3, \quad M_0 = 20, \quad (-1)20 + (7)3 = 1; \\
  m_1 = 4, \quad M_1 = 15, \quad (-1)15 + (4)4 = 1; \\
  m_2 = 5, \quad M_0 = 12, \quad (-2)12 + (5)5 = 1; 
  \]
- The integer \( c \) can be calculated as
  \[
  c = \sum_{i=0}^{N_i} c_i N_i, \quad (\text{mod } M) \\
  = (-20c_0 - 15c_1 - 24c_2) \pmod{60} 
  \]
- Example
  \[
  c = 17, \text{i.e.} (c_0, c_1, c_2) = (2, 1, 2) \\
  \text{Conversely, } c = (-20 \times 2 - 15 \times 1 - 24 \times 2) \pmod{60} = 17 
  \]

**Remarks**

- By taking residues, large integers are broken down into small pieces (that may be easy to add and multiply)
- Examples:
  \[
  7 \rightarrow (1, \quad 3, \quad 2 \quad ) \\
  +3 \rightarrow (0, \quad 3, \quad 3 \quad ) \\
  \overline{10 \rightarrow (1 \text{ mod } 3, \ 6 \text{ mod } 4, \ 5 \text{ mod } 5) = (1,2,0)} \\
  \]
  \[
  7 \rightarrow (1, \quad 3, \quad 2 \quad ) \\
  \times3 \rightarrow (0, \quad 3, \quad 3 \quad ) \\
  \overline{21 \rightarrow (0 \text{ mod } 3, \ 9 \text{ mod } 4, \ 6 \text{ mod } 5) = (0,1,1)} 
  \]
CRT for Polynomials (1)

- Given a set of polynomials \( m^{(0)}(x), m^{(1)}(x), \ldots, m^{(k)}(x) \), 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:

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

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

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

Chinese Remainder Theorem (2)

- Define \( M^{(0)}(x) = M(x)/ m^{(0)}(x) \), then \( \gcd(M^{(0)}(x),m^{(0)}(x)) = 1 \). So there exists polynomials \( N^{(0)}(x) \) and \( n^{(0)}(x) \) with \( \gcd(M^{(0)}(x),m^{(0)}(x)) = 1 = N^{(0)}(x)M^{(0)}(x) + n^{(0)}(x)m^{(0)}(x), \ i=0,1,\ldots,k \)

- The system of equations \( c^{(i)}(x) = c(x) \pmod{m^{(i)}(x)} \), \( 0 \leq i \leq k \), 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). $R_{-2} \left[ 5x^2 + 3x + 5 \right] = 5(-2)^2 + 3(-2) + 5 = 19$

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

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

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 $m(p) = m^{(0)}(p)m^{(1)}(p)...m^{(k)}(p)$. 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., $m(p) = m_1^{(i)}(p)m_2^{(i)}(p)\cdots m_k^{(i)}(p)$

2. Let $M^{(i)}(p) = m(p)/m^{(i)}_0(p)$. Use the Euclidean GCD algorithm to solve $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}_0(p) = 1$ for $N^{(i)}(p)$.

3. Compute: $h^{(i)}(p) = h(p) \mod m^{(i)}_0(p), x^{(i)}(p) = x(p) \mod m^{(i)}_0(p)$ for $i = 0,1,\cdots,k$

4. Compute: $s^{(i)}(p) = h^{(i)}(p)x^{(i)}(p) \mod m^{(i)}_0(p)$, for $i = 0,1,\cdots,k$

5. Compute $s(p)$ by using:
$$s(p) = \sum_{i=0}^k s^{(i)}(p)N^{(i)}(p)M^{(i)}(p) \mod m(p)$$

Example: $2\times3$ Winograd Algorithm

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

  
  \[
  w(p) = p(p-1)(p+1) 
  \]

- 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 $M^{(i)}(p) = w(p)/m^{(i)}_0(p)$ and $N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}_0(p) = 1$ for $i = 0,1,2$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$m^{(i)}(p)$</th>
<th>$M^{(i)}(p)$</th>
<th>$n^{(i)}(p)$</th>
<th>$N^{(i)}(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p$</td>
<td>$p^2-p^2+1$</td>
<td>$p+1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>1</td>
<td>$p-1$</td>
<td>$p^2+p$</td>
<td>$-\frac{1}{2}(p^2+p+2)$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>2</td>
<td>$p^2+1$</td>
<td>$p^2-p$</td>
<td>$-\frac{1}{2}(p-2)$</td>
<td>$\frac{1}{2}(p-1)$</td>
</tr>
</tbody>
</table>

- Compute residues from $h(p) = h_0 + h_1p$, $x(p) = x_0 + x_1p + x_2p^2$

  
  \[
  h^{(0)}(p) = h_0, \quad x^{(0)}(p) = x_0 
  \]
  \[
  h^{(1)}(p) = h_0 + h_1, \quad x^{(1)}(p) = x_0 + x_1 + x_2 
  \]
  \[
  h^{(2)}(p) = h_0 + h_1p, \quad x^{(2)}(p) = (x_0 - x_2) + x_1p 
  \]

This step requires no multiplication.
Example: 2×3 Winograd Algorithm

\[ s^{(0)}(p) = h_2x_6 = s_0^{(0)}, \quad s^{(1)}(p) = (h_0 + h_1)(x_0 + x_1 + x_2) = s_0^{(1)} \]

\[ s^{(2)}(p) = (h_0 + h_1)p((x_0 - x_1) + x_1p) \mod (p^2 + 1) = h_0(x_0 - x_1) - h_1x_1 + (h_0x_1 + h_1(x_0 - x_1))p = s_0^{(2)} + s_1^{(2)}p \]

- 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:

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

- Then:

\[
s(p) = \sum_{p^0}^{2} s^{(0)}(p)N^{(0)}(p)M^{(0)}(p) \mod m(p)
\]

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

Example: 2×3 Winograd Algorithm

- Substitute \( s^{(0)}(p) \), \( s^{(1)}(p) \), \( s^{(2)}(p) \) into \( s(p) \) to obtain the following table:

<table>
<thead>
<tr>
<th></th>
<th>( p^0 )</th>
<th>( p^1 )</th>
<th>( p^2 )</th>
<th>( p^3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0^{(0)} )</td>
<td>( -s_0^{(0)} )</td>
<td>( s_0^{(0)} )</td>
<td>( -s_0^{(0)} )</td>
<td></td>
</tr>
<tr>
<td>( \frac{1}{2}s_0^{(1)} )</td>
<td>0</td>
<td>( \frac{1}{2}s_0^{(1)} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \frac{1}{2}s_0^{(2)} )</td>
<td>( -s_0^{(2)} )</td>
<td>( \frac{1}{2}s_0^{(2)} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( \frac{1}{2}s_1^{(2)} )</td>
<td>0</td>
<td>( -\frac{1}{2}s_1^{(2)} )</td>
<td></td>
</tr>
</tbody>
</table>

- Therefore, we have

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

VSP Lecture6 - Fast Algorithms for DSP (cwliu@twins.ee.nctu.edu.tw) 3-30
Example: $2 \times 3$ Winograd Algorithm

Notice that

\[
\begin{bmatrix}
\frac{1}{2} s_0 \\
\frac{1}{2} s_0 \frac{1}{2} s_1 \\
\frac{1}{2} s_0 \frac{1}{2} s_1 \frac{1}{2} s_2
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
h_0 & 0 & 0 & 0 & 0 \\
0 & \frac{h_0 + h_1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{h_0 + h_2}{2} & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2
\end{bmatrix}
\]

So, finally we have:

\[
\begin{bmatrix}
s_0 \\
\frac{1}{2} s_0 \\
\frac{1}{2} s_0 \frac{1}{2} s_1 \\
\frac{1}{2} s_0 \frac{1}{2} s_1 \frac{1}{2} s_2
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 2 & 1 & -1 & 0 \\
1 & 0 & -2 & 0 & 2 & 0 \\
-1 & 0 & 0 & -1 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
h_0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{h_0 + h_1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{h_0 + h_2}{2} & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2
\end{bmatrix}
\]

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 $s'(p) = s(p) - h_0 x_{N-1} m(p)$. Note that $s'(p) \pmod{m(p)} = s(p) \pmod{m(p)}$.
- Modified Winograd Algorithm:
  - Choose $m(p)$ with a degree equal to that of $s(p)$
  - Apply CRT to $s(p)$
  - $s(p) = s(p) + h_0 x_{N-1} m(p)$. 

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

- $4 \times 4$ linear convolution algorithm
  - Let $h(p) = h_0 + h_1 p + h_2 p^2 + h_3 p^3$. $x(p) = x_0 + x_1 p + x_2 p^2 + x_3 p^3$ and $s(p) = h(p) x(p)$.
  - First, we need to decompose the $4 \times 4$ convolution into a $2 \times 2$ convolution.
  - Define $h'_0(p) = h_0 + h_1 p$, $h'_1(p) = h_2 + h_3 p$.
  - $x'_0(p) = x_0 + x_1 p$, $x'_1(p) = x_2 + x_3 p$.
  - Then, we have: Polyphase decomposition.
  - $h(p) = h'_0(p) + h'_1(p) q$.
  - $x(p) = x'_0(p) + x'_1(p) q$.
  - $s(p) = h(p) x(p) = h'_0(p) x'_0(p) + h'_1(p) x'_1(p) q$.
  - $s'_0(p) = s'_0(p) + s'_1(p) q + s'_2(p) q^2 = s(p, q)$. 

\[
\begin{align*}
\text{Example} & \quad 4 \times 4 \text{ linear convolution algorithm} \\
& \quad \text{Let } h(p) = h_0 + h_1 p + h_2 p^2 + h_3 p^3. \quad x(p) = x_0 + x_1 p + x_2 p^2 + x_3 p^3 \\
& \quad \text{and } s(p) = h(p) x(p). \\
& \quad \text{First, we need to decompose the } 4 \times 4 \text{ convolution into a } 2 \times 2 \text{ convolution.} \\
& \quad \text{Define } h'_0(p) = h_0 + h_1 p, \quad h'_1(p) = h_2 + h_3 p. \\
& \quad x'_0(p) = x_0 + x_1 p, \quad x'_1(p) = x_2 + x_3 p. \\
& \quad \text{Then, we have: Polyphase decomposition.} \\
& \quad h(p) = h'_0(p) + h'_1(p) q. \\
& \quad x(p) = x'_0(p) + x'_1(p) q. \\
& \quad s(p) = h(p) x(p) = h'_0(p) x'_0(p) + h'_1(p) x'_1(p) q. \\
& \quad s'_0(p) = s'_0(p) + s'_1(p) q + s'_2(p) q^2 = s(p, q). \\
\end{align*}
\]
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:

\[
\begin{bmatrix}
  s_0'(p) \\
  s_1'(p) \\
  s_2'(p)
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 \\
  1 & -1 & 1 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  h_0'(p) \\
  h_1'(p) - h_2'(p) \\
  k'(p)
\end{bmatrix}
\begin{bmatrix}
  1 & 0 \\
  1 & -1 \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  x_0'(p) \\
  x_1'(p) \\
  x_2'(p)
\end{bmatrix}
\]

- 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

Linear Shift

Linear Shift

Linear Shift
Circular Shift

- Conventional shift (linear shift)

Circular Convolution

- Given two sequences $x_i$ and $h_i$, $0 \leq i \leq n-1$, of block length $n$
- Notation: $((n-k)) = n-k \mod n$
- Cyclic (or circular) convolution $s'_i$, $0 \leq i \leq n-1$, 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:

$$s'(p) = s(p) \mod p^n - 1 = h(p)x(p) \mod p^n - 1$$
Direct Implementation

- Consider the following system

\[ x \cdot O_h \rightarrow s \]

\( n \)-point sequence \( n \)-point sequence \( n \)-point sequence

4x4 cyclic convolution

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

16 multiplications
12 additions

Circular Convolution

Circular shift

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

<table>
<thead>
<tr>
<th>i</th>
<th>m^{(0)}(p)</th>
<th>M^{(0)}(p)</th>
<th>n^{(0)}(p)</th>
<th>N^{(0)}(p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p-1</td>
<td>p^3 + p^2 + p - 1</td>
<td>-\frac{1}{4}(p^2 + 2p + 3)</td>
<td>\frac{1}{4}</td>
</tr>
<tr>
<td>1</td>
<td>p+1</td>
<td>p^3 - p^2 + p - 1</td>
<td>\frac{1}{4}(p^2 - 2p + 3)</td>
<td>-\frac{1}{4}</td>
</tr>
<tr>
<td>2</td>
<td>p^2 + 1</td>
<td>p^2 - 1</td>
<td>\frac{1}{2}</td>
<td>-\frac{1}{2}</td>
</tr>
</tbody>
</table>

- Compute the residues
  - \hat{h}^{(0)}(p) = h_0 + h_1 + h_2 + h_3 = h_0^{(0)}, \quad p=1
  - \hat{h}^{(1)}(p) = h_0 - h_1 + h_2 - h_3 = h_0^{(1)}, \quad p=-1
  - h^{(2)}(p) = (h_0 - h_1) + (h_1 - h_3)p = h_0^{(2)} + h_1^{(2)}p, \quad p^2 = 1

\begin{align*}
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 \\
s^{(0)}(p) &= \hat{h}^{(0)}(p) \cdot x^{(0)}(p) = h_0^{(0)} \cdot x_0^{(0)} = s_0^{(0)}; \\
s^{(1)}(p) &= \hat{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 = [\hat{h}^{(2)}(p) \cdot x^{(2)}(p)] \mod (p^2 + 1)
\end{align*}

- Since
  - s_0^{(2)} = h_0^{(2)} x_0^{(2)} - h_1^{(2)} x_1^{(2)} = h_0^{(2)} (x_0^{(2)} + x_1^{(2)}) - h_1^{(2)} [x_0^{(2)} + h_1^{(2)} x_0^{(2)}] \\
  - s_1^{(2)} = h_0^{(2)} x_1^{(2)} + h_1^{(2)} x_0^{(2)} = h_0^{(2)} (x_0^{(2)} + x_1^{(2)}) + h_1^{(2)} [h_0^{(2)} - h_1^{(2)} x_0^{(2)}] \\

- or in matrix-form

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

- Computations so far require 5 multiplications
Example

- Notice that:

\[
\begin{bmatrix}
\frac{1}{4} s_0^{(0)} \\
\frac{1}{4} x_0^{(1)} \\
\frac{1}{2} s_0^{(2)} \\
\frac{1}{2} x_0^{(2)}
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix} \cdot \begin{bmatrix}
x_0^{(0)} \\
x_0^{(1)} \\
x_0^{(2)} \\
x_0^{(2)} + x_0^{(2)}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{1}{4} h_0^{(0)} \\
\frac{1}{4} h_0^{(1)} \\
0 \\
0
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
x_0^{(0)} \\
x_0^{(1)} \\
x_0^{(2)} \\
x_0^{(2)}
\end{bmatrix}
\]
Therefore, we have

\[
\begin{bmatrix}
    s_0 \\ s_1 \\ s_2 \\ s_3 \\
\end{bmatrix} =
\begin{bmatrix}
    1 & 1 & 1 & 0 & -1 \\
    1 & -1 & 1 & 1 & 0 \\
    1 & 1 & -1 & 0 & 1 \\
    1 & -1 & -1 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
    \frac{b_3 + b_4 + b_5 + b_6}{4} & 0 & 0 & 0 & 0 \\
    \frac{b_3 - b_4 + b_5 - b_6}{4} & 0 & 0 & 0 & 0 \\
    0 & \frac{b_3 - b_4}{2} & 0 & 0 & 0 \\
    0 & 0 & \frac{-b_3 + b_4}{2} & 0 & 0 \\
\end{bmatrix}
\]

5 multiplications
15 additions

Discrete Fourier Transform

- Discrete Fourier transform (DFT) pairs

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

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

where \( W_N^{kn} = e^{-j \frac{2\pi kn}{N}} \)

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

\[
\begin{bmatrix}
    x_0 \\ x_1 \\ x_2 \\ x_3 \\
\end{bmatrix} =
\begin{bmatrix}
    1 & 1 & 1 & 1 \\
    1 & -1 & 1 & -1 \\
    1 & 1 & -1 & -1 \\
    1 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3 \\
\end{bmatrix}
\]

N complex multiplications
N-1 complex additions

2\pi/N
Decimation in Time

\[ X_N[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad \rightarrow \quad N\text{-point DFT} \]

\[
= \sum_{n \text{ even}}^{N/2-1} x[2l] W_N^{2lk} + \sum_{n \text{ odd}}^{N/2-1} x[2l+1] W_N^{(2l+1)k}
\]

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

The number of complex multiplications is \(N/2 \cdot N + 2N/2 = N^2 - N\) for the DFT of length \(N\).

\[ \text{two } N/2 - \text{point DFT's}! \]

Using a briefer system of notation:

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

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!
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)

\[
W_N^{N/2} = W_N^N e^{j\pi} = -W_N^{N/2}
\]

2 complex multiplications
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

Stage 1

Stage 2

Stage 3
8-point FFT

Normal order v.s. 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 $\Rightarrow$ only one storage array will be required

Decimation in Frequency (DIF)

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

  $$X[k] = \begin{cases} X[2r] & 0 \leq r \leq \frac{N}{2} - 1 \\ X[2r+1] & \end{cases}$$

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

  $$W_N^{2rn} = W_N^{2r} W_N^n = W_N^{2r}$$

Substitution of variables
DIF FFT Algorithm (1)

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

is just \( N/2 \)-point DFT. Similarly,

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

\( v = \log_2 N \) 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.

\[ W_N^0 \quad W_N^1 \]

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, \( a, b, c, d, e, f \in \mathbb{R} \)
  - The direct implementation requires 4 multiplications and 2 additions
  - However, the number of multiplication can be reduced to 3 at the expense of 3 extra additions by using the identities
    \[
    \begin{align*}
    ac - bd &= a(c - d) + d(a - b) & \quad 3 \text{ multiplications} \\
    ad + bc &= b(c + d) + d(a - b) & \quad 5 \text{ additions}
    \end{align*}
    \]
Complex Multiplication

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

→ Speed?, Area?, Power? …

FIR Filters

\[
x(1), x(2), x(3), \ldots \rightarrow H(z) \rightarrow y(n) = h(n) \ast x(n) \Leftrightarrow Y(z) = H(z)X(z)
\]

Transform-domain

Time-domain

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

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

\[
h(n) \ast x(n) = \sum_{i=0}^{N-1} h(i)x(n-i), \quad n = 0, 1, 2, \ldots, \infty
\]
**Example: Linear Phase FIR**

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

\[
\]

By exploiting substructure sharing to reduce area.

**An Efficient Decomposition**

- **Example: 2-fold decomposition**
  \[
  \]

- **Example 3-fold decomposition**
  \[
  \]

- **General case (N-fold decomposition)**
  \[
  H(z) = \sum_{k=-\infty}^{\infty} h[k]z^{-k} = \sum_{l=0}^{N-1} z^{-l}H_i(z^N), \text{ where } H_i(z) = \sum_{k=-\infty}^{\infty} h[Nk+l]z^{-k}
  \]
Traditional Parallel Architecture

- 2-fold parallel architecture

\[ X(z) = X_0(z^{-1}) + z^{-1}X_1(z^{-1}) \]
\[ H(z) = H_0(z^{-1}) + z^{-1}H_1(z^{-1}) \]

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

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

Traditional Parallel FIR

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

L-parallel FIR filter of length N/L requires
1. \( L^2 (N/L) \) multiplications, i.e. LN multiplications
2. \( L^2 (N/L - 1) \) additions, i.e. L(N-1) additions

~ LN multiply-add operations
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 $N/L$

\[ \Rightarrow \text{It requires } 2N-N/L \text{ multiplications} \]