Chapter 8: Efficient Computation of the DFT: FFT Algorithms

Discrete Fourier Transform (DFT) Pair

- DFT Pair:
  \[ X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}, \quad k = 0, 1, \ldots, N - 1 \]
  \[ x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi k \frac{n}{N}}, \quad n = 0, 1, \ldots, N - 1 \]

- The IDFT can be computed using the DFT by
  1. reversing the order of the input to the DFT
  2. scaling the associated output by \(\frac{1}{N}\)

- The complexity of the IDFT is the same as the complexity of the DFT.

Computation of the IDFT via the DFT

\[ X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}} \]
\[ x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi k \frac{n}{N}}, \quad \text{let } k' = n \text{ and } n' = k \]
\[ x(k') = \frac{1}{N} \sum_{n'=0}^{N-1} X(n') e^{j2\pi n' \frac{k'}{N}} \]
\[ = \frac{1}{N} \sum_{n''=0}^{N-1} X(n') e^{j2\pi k' \frac{n''}{N}} \quad \text{let } n'' = -n' \]
Computing the IDFT via the DFT

**DFT Pair**

- **DFT Pair:**
  \[
  X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, \quad k = 0, 1, \ldots, N-1
  \]
  \[
  x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N}, \quad n = 0, 1, \ldots, N-1
  \]

- **New notation:** \( W_N = e^{-j2\pi/N} \)
  \[
  X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad k = 0, 1, \ldots, N-1
  \]
  \[
  x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}, \quad n = 0, 1, \ldots, N-1
  \]

The IDFT can be computed using the DFT by
1. reversing the order of the input to the DFT
2. scaling the associated output by \( \frac{1}{N} \)

- The complexity of the IDFT is the same as the complexity of the DFT.

**Complexity of the DFT (and IDFT)**

- **Straightforward implementation of DFT to compute** \( X(k) \) **for** \( k = 0, 1, \ldots, N-1 \) **requires:**
  - \( N^2 \) complex multiplications
    - 1 complex mult = \( (a_R + ja_I) \cdot (b_R + jb_I) = (a_R \cdot b_R - a_I \cdot b_I) + j(a_R \cdot b_I + a_I \cdot b_R) \)
    - 4 real mult + 2 real add
  - \( 4N^2 \) real multiplications
  - \( N(N-1) \) complex additions
    - 1 complex add = \( (a_R + ja_I) + (b_R + jb_I) = (a_R + b_R) + j(a_I + b_I) = 2 \) real add
    - \( 2N(N-1) \) + \( 2N^2 \) (from complex mult) real additions
      = \( 2N(2N-1) \) real additions.
Chapter 8: Efficient Computation of the DFT: FFT Algorithms

8.1 FFT Algorithms

Complexity of the DFT

▶ Is $O(N^2)$ high?

▶ Yes. A linear increase in the length of the DFT increases the complexity by a power of two.
▶ Given the multitude of applications where Fourier analysis is employed (linear filtering, correlation analysis, spectrum analysis), a method of efficient computation is needed.

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms 9 / 42

Chapter 8: Efficient Computation of the DFT: FFT Algorithms

Complexity of the DFT

▶ How can we reduce complexity?

▶ Exploit symmetry of the complex exponential.

\[
W_N^{k+N} = W_N^k
\]
\[
\text{LHS} = W_N^{k+N} = e^{-j2\pi \frac{k+N}{N}} = e^{-j2\pi \frac{k}{N}} e^{-j2\pi \frac{N}{N}} = e^{-j2\pi \frac{k}{N}} e^{-j\pi} = e^{-j2\pi \frac{k}{N}} (\cos(-\pi) + j \sin(-\pi)) = -e^{-j2\pi \frac{k}{N}} = \text{RHS}
\]

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms 10 / 42

Chapter 8: Efficient Computation of the DFT: FFT Algorithms

Complexity of the DFT

▶ How can we reduce complexity?

▶ Exploit periodicity of the complex exponential.

\[
W_N^{k+N} = W_N^k
\]
\[
\text{LHS} = W_N^{k+N} = e^{-j2\pi \frac{k+N}{N}} = e^{-j2\pi \frac{k}{N}} e^{-j2\pi \frac{N}{N}} = e^{-j2\pi \frac{k}{N}} e^{-j2\pi} = e^{-j2\pi \frac{k}{N}} \cdot (\cos(-2\pi) + j \sin(-2\pi)) = e^{-j2\pi \frac{k}{N}} (1) = e^{-j2\pi \frac{k}{N}} = W_N^k = \text{RHS}
\]

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms 11 / 42

Chapter 8: Efficient Computation of the DFT: FFT Algorithms

Divide-and-Conquer for Complexity Reduction

▶ Consider $N = L \cdot M$ where $N, L, M \in \mathbb{Z}^+$

▶ If the length of a signal is prime, then we can zero pad the signal so that $N$ is not prime.

▶ Decompose $N$-point DFT into successfully smaller DFTs

▶ $M L$-point DFTs + $L M$-point DFTs

▶ Resulting algorithms are known as fast Fourier transform (FFT) algorithms

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms 12 / 42
Divide-and-Conquer for Complexity Reduction

<table>
<thead>
<tr>
<th>Naive DFT</th>
<th>Divide-and-Conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(N^2)$</td>
<td>$O(L^2M + LM^2) = O(LN + MN)$</td>
</tr>
</tbody>
</table>

Example: Let $N = 500$

<table>
<thead>
<tr>
<th></th>
<th>Cmplx Mult</th>
<th>Cmplx Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 500$ Direct</td>
<td>250,000</td>
<td>249,500</td>
</tr>
<tr>
<td>$L = 2$ and $M = 250$</td>
<td>126,500</td>
<td>125,000</td>
</tr>
<tr>
<td>$L = 5$ and $M = 100$</td>
<td>53,000</td>
<td>51,500</td>
</tr>
<tr>
<td>$L = 20$ and $M = 25$</td>
<td>23,000</td>
<td>21,500</td>
</tr>
</tbody>
</table>

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms

Divide-and-Conquer for Complexity Reduction

- **WLOG** suppose $x(n)$ is stored column-wise and $X(k)$ is stored row-wise.
- $n = mL + l$ and $k = Mp + q$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

$$X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) W_N^{(Mp+q)(mL+l)}$$

$$W_N^{(Mp+q)(mL+l)} = W_N^{MLmp} W_N^{mlq} W_N^{Mpl} W_N^{lq}$$

Divide-and-Conquer for Complexity Reduction

- **Common 1-D to 2-D mappings** for storage of $x(n)$:
  - $n = Ml + m$ (values stored row-wise)
  - $n = l + mL$ (values stored column-wise)

- **Common 1-D to 2-D mappings** for storage of $X(k)$:
  - $k = Mp + q$ (values stored row-wise)
  - $k = p + qL$ (values stored column-wise)

$$X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) W_N^{(Mp+q)(mL+l)}$$

$$W_N^{MLmp} = e^{-j\frac{2\pi}{LM}mp} = e^{-j\frac{\pi}{2}mp} = 1$$

$$W_N^{mlq} = e^{-j\frac{\pi}{2}mLq} = e^{-j\frac{\pi}{2}mq} = W_N^{mq} W_N^{m/L} = W_M^{mq}$$

$$W_N^{Mpl} = e^{-j\frac{\pi}{2}Mpl} = e^{-j\frac{\pi}{2}pl} = W_L^{pl}$$

$$X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) \cdot 1 \cdot W_N^{mlq} W_L^{pl} W_N^{lq}$$

Dr. Deepa Kundur (University of Toronto) Efficient Computation of the DFT: FFT Algorithms
Divide-and-Conquer for Complexity Reduction

\[ X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) \cdot 1 \cdot W_M^{mq} W_L^{pl} W_N^{lq} \]

\[ = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[ \sum_{m=0}^{M-1} x(l, m) W_M^{mq} \right] \right\} W_L^{pl} \]

\[ = \sum_{l=0}^{L-1} \left\{ \sum_{m=0}^{M-1} x(l, m) W_M^{mq} \right\} \cdot W_N^{lq} \cdot W_L^{pl} \]

\[ \text{M-DFT} \]

\[ \text{L-DFT} \]

Complexity Comparison

1. Compute M-DFTs:
   \( LM^2 \) complex mult and \( LM(M - 1) \) complex add

2. Compute a new rectangular array \( G(l, q) \):
   \( LM \) complex mult and 0 complex add

3. Compute the L-DFTs:
   \( ML^2 \) complex mult and \( ML(L - 1) \) complex add

Therefore, for divide-and-conquer:

- Complex multiplications: \( N(M + L + 1) \)
- Complex additions: \( N(M + L - 2) \)

Compared to complexity of direct \( N \)-DFT:

- Complex multiplications: \( N^2 \)
- Complex additions: \( N(N - 1) \)

Example: Let \( N = 500 \).

- Direct \( N \)-DFT:
  - Complex multiplications: 250,000
  - Complex additions: 249,500

- Divide-and-conquer for \( L = 2 \) and \( M = 250 \):
  - Complex multiplications: 126,500
  - Complex additions: 125,000

- Divide-and-conquer for \( L = 5 \) and \( M = 100 \):
  - Complex multiplications: 53,000
  - Complex additions: 51,500

- Divide-and-conquer for \( L = 20 \) and \( M = 25 \):
  - Complex multiplications: 23,000
  - Complex additions: 21,500
**Divide-and-Conquer Algorithm 1**

For \( n = mL + l \) and \( k = Mp + q \):

\[
X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) W_N^{(Mp+q)(mL+l)} = \sum_{l=0}^{L-1} \left\{ W_N^m \sum_{m=0}^{M-1} x(l, m) W_M^{mq} \right\} W_L^{pl}
\]

1. Store the signal column-wise.
2. Compute the \( M \)-DFT of each row.
3. Multiply the resulting array by the phase factors \( W_N^m \).
4. Compute the \( L \)-DFT of each column.
5. Read the resulting array row-wise.

**Divide-and-Conquer Algorithm 2**

For \( n = Ml + m \) and \( k = qL + p \):

\[
X(p, q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l, m) W_N^{(qL+p)(Ml+m)} = \sum_{m=0}^{M-1} \left\{ W_N^{mp} \sum_{l=0}^{L-1} x(l, m) W_L^{lp} \right\} W_M^{mq}
\]

1. Store the signal row-wise.
2. Compute the \( L \)-DFT of each column.
3. Multiply the resulting array by the phase factors \( W_N^{mp} \).
4. Compute the \( M \)-DFT of each row.
5. Read the resulting array column-wise.

**Divide-and-Conquer Implementation Example**

Algorithm 1:

See *Figure 8.1.3 of text*.

**Divide-and-Conquer for Complexity Reduction**

- Therefore for \( N = L \cdot M \) where \( N, L, M \in \mathbb{Z}^+ \) we can decompose \( N \)-point DFT into successfully smaller DFTs:
  - \( M \) \( L \)-point DFTs AND \( L \) \( M \)-point DFTs

<table>
<thead>
<tr>
<th>Naive DFT</th>
<th>Divide-and-Conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(N^2) )</td>
<td>( O(L^2 M + L M^2) = O(LN + MN) )</td>
</tr>
</tbody>
</table>
Radix-2 FFT Algorithms

- For $N = 2^v$,
  - Divide-and-conquer results in the repeated use of $r$-DFTs that form a regular pattern and reduce complexity.
  - called radix-$r$ FFT algorithms

- Consider $r = 2$ (most widely used algorithms):
  - decimation-in-time algorithm
  - decimation-in-frequency algorithm

Radix-2 FFT: Decimation-in-time

Note: $W_N^2 = e^{-j2\pi/2} = e^{-j\pi} = W_{N/2}$

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = \sum_{n=0}^{N-1} x(n)W_N^{kn}$$

$$= \sum_{m=0}^{(N/2)-1} x(2m)W_N^{2km} + \sum_{m=0}^{(N/2)-1} x(2m+1)W_N^{2km}$$

$$= \sum_{m=0}^{(N/2)-1} f_1(m)W_N^{km} + \sum_{m=0}^{(N/2)-1} f_2(m)W_N^{km}$$

$$= F_1(k) + W_N^kF_2(k), \quad k = 0, 1, \ldots, N - 1$$

Radix-2 FFT: Decimation-in-time

Note: since $F_1(k)$ and $F_2(k)$ are $N/2$-DFTs:

$$F_1(k) = F_1(k + \frac{N}{2})$$

$$F_2(k) = F_2(k + \frac{N}{2})$$

we have,

$$X(k) = F_1(k) + W_N^kF_2(k)$$

$$X(k + \frac{N}{2}) = F_1(k + \frac{N}{2}) + W_N^{k+\frac{N}{2}}F_2(k + \frac{N}{2})$$

$$= F_1(k) - W_N^kF_2(k)$$

since $W_N^{k+\frac{N}{2}} = e^{-j\frac{2\pi}{N}k} = e^{-j\frac{\pi}{N}k} = e^{-j\frac{\pi}{N}(-1)} = -W_N^k$
Radix-2 FFT: Decimation-in-time

Therefore,

\[ X(k) = F_1(k) + W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]
\[ X(k + \frac{N}{2}) = F_1(k) - W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]

See \( \bullet \) Figure 8.1.4 of text.

Radix-2 FFT: Decimation-in-frequency

Repeating the decimation-in-time for \( f_1(n) \) and \( f_2(n) \), we obtain:

\[ v_{11}(n) = f_1(2n) \quad n = 0, 1, \ldots, N/4 - 1 \]
\[ v_{12}(n) = f_1(2n + 1) \quad n = 0, 1, \ldots, N/4 - 1 \]
\[ v_{21}(n) = f_2(2n) \quad n = 0, 1, \ldots, N/4 - 1 \]
\[ v_{22}(n) = f_2(2n + 1) \quad n = 0, 1, \ldots, N/4 - 1 \]

and

\[ F_1(k) = V_{11}(k) + W_{N/2}^k V_{12}(k) \quad k = 0, 1, \ldots, N/4 - 1 \]
\[ F_1(k + N/4) = V_{11}(k) - W_{N/2}^k V_{12}(k) \quad k = 0, 1, \ldots, N/4 - 1 \]
\[ F_2(k) = V_{21}(k) + W_{N/2}^k V_{22}(k) \quad k = 0, 1, \ldots, N/4 - 1 \]
\[ F_2(k + N/4) = V_{21}(k) - W_{N/2}^k V_{22}(k) \quad k = 0, 1, \ldots, N/4 - 1 \]

consisting of \( N/4 \)-DFTs.

Radix-2 FFT: Decimation-in-frequency

For \( N = 8 \):

- See \( \bullet \) Figure 8.1.5 of text.
- See \( \bullet \) Figure 8.1.6 of text.
- See \( \bullet \) Figure 8.1.7 of text.

Complexity:

- Each butterfly requires:
  - one complex multiplication
  - two complex additions
- In total, there are:
  - \( \frac{N}{2} \) butterflies per stage
  - \( \log N \) stages
- Order of the overall DFT computation is: \( O(N \log N) \)
Radix-2 FFT: Decimation-in-frequency

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

\[ = \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_N^k \sum_{n'=0}^{(N/2)-1} x(n' + \frac{N}{2}) W_N^{kn'} \]

\[ = \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + (W_N^k)^{N/2} \sum_{n'=0}^{(N/2)-1} x(n' + \frac{N}{2}) W_N^{kn'} \]

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

\[ = \sum_{n=0}^{(N/2)-1} \left[ x(n) + (-1)^k x \left( n + \frac{N}{2} \right) \right] W_N^{kn} \]

Therefore,

\[ X(2k) = \sum_{n=0}^{(N/2)-1} g_1(n) W_N^{kn/2} \]

\[ X(2k + 1) = \sum_{n=0}^{(N/2)-1} g_2(n) W_N^{kn/2} \]

Example: \( N = 8 \). See Figure 8.1.9 of text.

- Repeating the procedure for the remaining 4-DFTs, we obtain:
  - See Figure 8.1.11 of text.
  - The basic butterfly configuration is given by:
    - See Figure 8.1.10 of text. (Note: the “2” is a typo in the text for \((a - 2b)\))

Radix-2 FFT: Decimation-in-frequency

\[ X(k) = \sum_{n=0}^{(N/2)-1} x(n) (-1)^k x \left( n + \frac{N}{2} \right) W_N^{kn}, \; k = 0, \ldots, N \]

\[ X(2k) = \sum_{n=0}^{(N/2)-1} x(n) + (-1)^{2k} x \left( n + \frac{N}{2} \right) W_N^{2kn} \]

\[ = \sum_{n=0}^{(N/2)-1} x(n) \left[ x + x \left( n + \frac{N}{2} \right) W_N^{kn} \right]_{\text{for } k = 0, \ldots, N/2} \]

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

\[ = \sum_{n=0}^{(N/2)-1} x(n) \left[ x + x \left( n + \frac{N}{2} \right) W_N^{kn} \right]_{\text{for } k = 0, \ldots, N/2} \]

Complexity:

- Each butterfly requires:
  - one complex multiplication
  - two complex additions
- In total, there are:
  - \( \frac{N}{2} \) butterflies per stage
  - \( \log N \) stages
- Order of the overall DFT computation is: \( O(N \log N) \)
Other FFT algorithms exist such as the radix-4 algorithm and the split radix algorithms. Depending on the value of $N$, these can be more efficient. Most popular algorithms by far are the radix-2 decimation-in-time and radix-2 decimation-in-frequency algorithms.

### Convolution and Complexity

Let $x(n)$ and $h(n)$ be real signals.

Let the support of $x(n)$ be $n = 0, 1, \ldots, N - 1$. We are interested in determining $y(n)$ for $n = 0, 1, \ldots, N - 1$.

$$y(n) = x(n) \ast h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n - k)$$

$$= \sum_{k=0}^{N-1} x(k)h(n - k) \quad n = 0, 1, \ldots, N - 1$$

### Convolution using FFT

To compute the convolution of $x(n)$ (support: $n = 0, 1, \ldots, L - 1$) and $h(n)$ (support: $n = 0, 1, \ldots, M - 1$):

1. Assign $N = M + L - 1$.
2. Zero pad both $x(n)$ and $h(n)$ to have support $n = 0, 1, \ldots, N - 1$.
3. Take the $N$-FFT of $x(n)$ to give $X(k)$, $k = 0, 1, \ldots, N - 1$.
4. Take the $N$-FFT of $h(n)$ to give $H(k)$, $k = 0, 1, \ldots, N - 1$.
5. Produce $Y(k) = X(k) \cdot H(k)$, $k = 0, 1, \ldots, N - 1$.
6. Take the $N$-IFFT of $Y(k)$ to give $y(n)$, $n = 0, 1, \ldots, N - 1$. 

- $N$ real multiplications
- $N - 1$ real additions
- For all $n$ ($n=0, 1, \ldots, N-1$):
  - $N \cdot N = N^2 = O(N^2)$ real multiplications
  - $(N - 1) \cdot N = N(N - 1) = O(N^2)$ real additions
Convolution using FFT

To compute the convolution of \( x(n) \) (support: \( n = 0, 1, \ldots, L - 1 \)) and \( h(n) \) (support: \( n = 0, 1, \ldots, M - 1 \)):

1. Assign \( N = M + L - 1 \).
2. Zero pad both \( x(n) \) and \( h(n) \) to have support \( n = 0, 1, \ldots, N - 1 \).
   \( O(1) \)
3. Take the \( N \)-FFT of \( x(n) \) to give \( X(k) \), \( k = 0, 1, \ldots, N - 1 \).
   \( O(N \log N) \)
4. Take the \( N \)-FFT of \( h(n) \) to give \( H(k) \), \( k = 0, 1, \ldots, N - 1 \).
   \( O(N \log N) \)
5. Produce \( Y(k) = X(k) \cdot H(k) \), \( k = 0, 1, \ldots, N - 1 \).
   \( O(N) \)
6. Take the \( N \)-IFFT of \( Y(k) \) to give \( y(n) \), \( n = 0, 1, \ldots, N - 1 \).
   \( O(N \log N) \)

Complexity of Convolution using FFT

Therefore, the overall complexity of conducting convolution via the FFT is:

\[ O(N \log N) \]

which is lower than \( O(N^2) \) through direction computation of convolution in the time-domain. \( \blacksquare \)