Efficient Computation of the DFT: FFT Algorithms

Dr. Deepa Kundur

University of Toronto

Discrete-Time Signals and Systems

Reference:

Section 8.1 of

John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th edition, 2007.

Discrete Fourier Transform (DFT) Pair

- DFT Pair:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}}, \quad k = 0, 1, \ldots, N-1 \\ x(n) &= \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi k \frac{n}{N}}, \quad n = 0, 1, \ldots, N-1 \end{aligned}$$

- 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.

Computing the IDFT via the DFT

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) e^{-j2\pi k \frac{n}{N}} \\ x(n) &= \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi k \frac{n}{N}}, \quad \text{let } k' = n \text{ and } n' = k \\ x(k') &= \frac{1}{N} \sum_{n'=0}^{N-1} X(n') e^{j2\pi n' \frac{k'}{N}} \\ &= \frac{1}{N} \sum_{n'=0}^{N-1} X(n') e^{j2\pi k' \frac{n'}{N}} \quad \text{let } n'' = -n' \end{aligned}$$

Computing the IDFT via the DFT

$$
\begin{aligned}
x(k') &= \frac{1}{N}\sum_{n'=0}^{N-1} X(n')e^{j2\pi k'\frac{n'}{N}} \quad \text{let } n'' = -n' \\
&= \frac{1}{N}\sum_{n'=0}^{N-1} X_p(n')e^{j2\pi k'\frac{n'}{N}} = \frac{1}{N}\sum_{n''=-(N-1)}^{0} X_p(-n'')e^{-j2\pi k'\frac{n''}{N}} \\
&= \frac{1}{N}\sum_{n''=-(N-1)}^{0} \underbrace{X_p(-n'')e^{-j2\pi k'\frac{n''}{N}}}_{\text{periodic with period } N} \\
&= \frac{1}{N}\sum_{n''=-(N-1)+(N-1)}^{0+(N-1)} X_p(-n''+N)e^{-j2\pi k'\frac{n''}{N}}
\end{aligned}
$$

Computing the IDFT via the DFT

$$
\begin{aligned}
x(k) &= \frac{1}{N}\sum_{n=0}^{N-1} X_p(N-n)e^{-j2\pi k\frac{n}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} X(N-n)e^{-j2\pi k\frac{n}{N}} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} \underbrace{X(N-n)}_{\text{DFT argument}}\, e^{-j2\pi k\frac{n}{N}} \quad \text{where } X(N) \equiv X(0)
\end{aligned}
$$

- 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.

DFT Pair

- DFT Pair:

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)e^{-j2\pi k\frac{n}{N}}, \quad k = 0, 1, \ldots, N-1 \\
x(n) &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi k\frac{n}{N}}, \quad n = 0, 1, \ldots, N-1
\end{aligned}
$$

- New notation: $W_N = e^{-j\frac{2\pi}{N}}$

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{kn}, \quad k = 0, 1, \ldots, N-1 \\
x(n) &= \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-kn}, \quad n = 0, 1, \ldots, N-1
\end{aligned}
$$

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 = $(a_R + ja_I)\times(b_R + jb_I) = (a_R \times b_R - a_I \times b_I) + j(a_R \times b_I + a_I \times b_R)$ = 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.

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.

Complexity of the DFT

- How can we reduce complexity?
  - Exploit symmetry of the complex exponential.

$$\begin{aligned}
W_N^{k+\frac{N}{2}} &= -W_N^k \\
\text{LHS} = W_N^{k+\frac{N}{2}} &= e^{-j2\pi\frac{k+N/2}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j2\pi\frac{N/2}{N}} \\
&= e^{-j2\pi\frac{k}{N}} e^{-j\pi} \\
&= e^{-j2\pi\frac{k}{N}} \cdot (\cos(-\pi) + j\sin(-\pi)) \\
&= e^{-j2\pi\frac{k}{N}} (-1) \\
&= -e^{-j2\pi\frac{k}{N}} = -W_N^k = \text{RHS}
\end{aligned}$$

Complexity of the DFT

- How can we reduce complexity?
  - Exploit periodicity of the complex exponential.

$$\begin{aligned}
W_N^{k+N} &= W_N^k \\
\text{LHS} = W_N^{k+N} &= e^{-j2\pi\frac{k+N}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j2\pi\frac{N}{N}} \\
&= e^{-j2\pi\frac{k}{N}} e^{-j2\pi} \\
&= e^{-j2\pi\frac{k}{N}} \cdot (\cos(-2\pi) + j\sin(-2\pi)) \\
&= e^{-j2\pi\frac{k}{N}} (1) \\
&= e^{-j2\pi\frac{k}{N}} = W_N^k = \text{RHS}
\end{aligned}$$

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

Divide-and-Conquer for Complexity Reduction

Naive DFT	Divide-and-Conquer
$O(N^2)$	$O(L^2M + LM^2) = O(LN + MN)$

Example: Let $\boxed{N = 500}$

	Cmplx Mult	Cmplx Add
$N = 500$ Direct	250,000	249,500
$L = 2$ and $M = 250$	126,500	125,000
$L = 5$ and $M = 100$	53,000	51,500
$L = 20$ and $M = 25$	23,000	21,500

Divide-and-Conquer for Complexity Reduction

- See Figure 8.1.1 of text.
- Common 1-D to 2-D mappings for storage of $x(n)$:
  - $n = Ml + m$
    ($x(n)$ values stored row-wise)
  - $n = l + mL$
    ($x(n)$ values stored column-wised)
  - See Figure 8.1.2 of text.
- Common 1-D to 2-D mappings for storage of $X(k)$:
  - $k = Mp + q$
    ($X(k)$ values stored row-wise)
  - $k = p + qL$
    ($X(k)$ values stored column-wise)

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 $k = Mp + q$

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\ X(p,q) &= \sum_{m=0}^{M-1}\sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)} \\ W_N^{(Mp+q)(mL+l)} &= W_N^{MLmp} W_N^{mLq} W_N^{Mpl} W_N^{lq} \end{aligned}$$

Divide-and-Conquer for Complexity Reduction

$$\begin{aligned} X(p,q) &= \sum_{m=0}^{M-1}\sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)} \\ &= \sum_{m=0}^{M-1}\sum_{l=0}^{L-1} x(l,m) W_N^{MLmp} W_N^{mLq} W_N^{Mpl} W_N^{lq} \\ W_N^{MLmp} &= e^{-j\frac{2\pi}{N}MLmp} = e^{-j2\pi mp} = 1 \\ W_N^{mLq} &= e^{-j\frac{2\pi}{N}mLq} = e^{-j\frac{2\pi}{N/L}mq} = W_{N/L}^{mq} = W_M^{mq} \\ W_N^{Mpl} &= e^{-j\frac{2\pi}{N}Mpl} = e^{-j\frac{2\pi}{N/M}pl} = W_{N/M}^{pl} = W_L^{pl} \\ X(p,q) &= \sum_{m=0}^{M-1}\sum_{l=0}^{L-1} x(l,m) \cdot 1 \cdot W_M^{mq} W_L^{pl} W_N^{lq} \end{aligned}$$

Divide-and-Conquer for Complexity Reduction

$$\begin{aligned} X(p,q) &= \sum_{m=0}^{M-1}\sum_{l=0}^{L-1} x(l,m)\cdot 1 \cdot W_M^{mq} W_L^{pl} W_N^{lq} \\ &= \sum_{l=0}^{L-1}\left\{ W_N^{lq}\left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq}\right]\right\} W_L^{pl} \\ &= \underbrace{\sum_{l=0}^{L-1}\left\{ W_N^{lq}\underbrace{\left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq}\right]}_{M\text{-DFT}}\right\} W_L^{pl}}_{L\text{-DFT}} \end{aligned}$$

Divide-and-Conquer for Complexity Reduction

Steps to Compute $N(=ML)$-DFT:

1. Compute M-DFTs

$$F(l,q) = \sum_{m=0}^{M-1} x(l,m) W_M^{mq}, \quad 0 \le q \le M-1$$

for each of the rows $l = 0, 1, \ldots, L-1$.

2. Compute a new rectangular array $G(l,q)$ defined as

$$G(l,q) = W_N^{lq} F(l,q), \quad 0 \le l \le L-1, 0 \le q \le M-1$$

3. Compute the L-DFTs

$$X(p,q) = \sum_{l=0}^{L-1} G(l,q) W_L^{lp}$$

for each column $q = 0, 1, \ldots, M-1$, of the array $G(l,q)$.

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: $N(N-1)$

Complexity Comparison

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 $k = Mp + q$:

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

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^{lq}.
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 $k = qL + p$:

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

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^{pm}.
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

Naive DFT	**Divide-and-Conquer**
$O(N^2)$	$O(L^2M + LM^2) = O(LN + MN)$

Radix-r FFT Algorithms

- For $N = r^v$,
  - 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

- Decimation-in-time algorithm
  - equivalent to divide-and-conquer for $L = 2$ and $M = \frac{N}{2}$

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad k = 0, 1, \ldots, N-1 \\
&= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn} \\
&= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{k(2m)} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)} \\
&= \sum_{m=0}^{(N/2)-1} \underbrace{x(2m)}_{\equiv f_1(m)} W_N^{2km} + \sum_{m=0}^{(N/2)-1} \underbrace{x(2m+1)}_{\equiv f_2(m)} W_N^{2km} W_N^k
\end{aligned}
$$

Radix-2 FFT: Decimation-in-time

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

$$
\begin{aligned}
X(k) &= \sum_{m=0}^{(N/2)-1} \underbrace{x(2m)}_{\equiv f_1(m)} W_N^{2km} + \sum_{m=0}^{(N/2)-1} \underbrace{x(2m+1)}_{\equiv f_2(m)} W_N^{2km} W_N^k \\
&= \underbrace{\sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km}}_{\frac{N}{2}\text{–DFT of } f_1(m)} + W_N^k \underbrace{\sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km}}_{\frac{N}{2}\text{–DFT of } f_2(m)} \\
&= F_1(k) + W_N^k F_2(k), \quad k = 0, 1, \ldots, N-1
\end{aligned}
$$

Radix-2 FFT: Decimation-in-time

Note: since $F_1(k)$ and $F_2(k)$ are $\frac{N}{2}$-DFTs:

$$
\begin{aligned}
F_1(k) &= F_1(k + \frac{N}{2}) \\
F_2(k) &= F_2(k + \frac{N}{2})
\end{aligned}
$$

we have,

$$
\begin{aligned}
X(k) &= F_1(k) + W_N^k F_2(k) \\
X(k + \frac{N}{2}) &= F_1(k + \frac{N}{2}) + W_N^{k+\frac{N}{2}} F_2(k + \frac{N}{2}) \\
&= F_1(k) - W_N^k F_2(k)
\end{aligned}
$$

since $W_N^{k+\frac{N}{2}} = e^{-j\frac{2\pi}{N}(k+\frac{N}{2})} = e^{-j\frac{2\pi}{N}k} \cdot e^{-j\frac{2\pi}{N}\frac{N}{2}} = e^{-j\frac{2\pi}{N}k}(-1) = -W_N^k$

Radix-2 FFT: Decimation-in-time

Therefore,

$$
\begin{aligned}
X(k) &= F_1(k) + W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \\
X(k + \frac{N}{2}) &= F_1(k) - W_N^k F_2(k) \quad k = 0, 1, \ldots, \frac{N}{2} - 1
\end{aligned}
$$

See ‣ Figure 8.1.4 of text.

Radix-2 FFT: Decimation-in-time

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

$$
\begin{aligned}
v_{11}(n) &= f_1(2n) \quad n = 0, 1, \ldots, N/4 - 1 \\
v_{12}(n) &= f_1(2n+1) \quad n = 0, 1, \ldots, N/4 - 1 \\
v_{21}(n) &= f_2(2n) \quad n = 0, 1, \ldots, N/4 - 1 \\
v_{22}(n) &= f_2(2n+1) \quad n = 0, 1, \ldots, N/4 - 1
\end{aligned}
$$

and

$$
\begin{aligned}
F_1(k) &= V_{11}(k) + W_{N/2}^k V_{12}(k) \quad k = 0, 1, \ldots, N/4 - 1 \\
F_1(k + N/4) &= V_{11}(k) - W_{N/2}^k V_{12}(k) \quad k = 0, 1, \ldots, N/4 - 1 \\
F_2(k) &= V_{21}(k) + W_{N/2}^k V_{22}(k) \quad k = 0, 1, \ldots, N/4 - 1 \\
F_2(k + N/4) &= V_{21}(k) - W_{N/2}^k V_{22}(k) \quad k = 0, 1, \ldots, N/4 - 1
\end{aligned}
$$

consisting of $N/4$-DFTs.

Radix-2 FFT: Decimation-in-time

For $N = 8$.

- See ‣ Figure 8.1.5 of text.
- See ‣ Figure 8.1.6 of text.
- See ‣ 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

- Decimation-in-frequency algorithm
  - equivalent to divide-and-conquer for $L = \frac{N}{2}$ and $M = 2$

$$
\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad k = 0, 1, \ldots, N-1 \\
&= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn} \quad \text{let } n' = n - \tfrac{N}{2} \\
&= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n'=0}^{(N/2)-1} x\left(n' + \frac{N}{2}\right) W_N^{k(n'+\frac{N}{2})} \\
&= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n'=0}^{(N/2)-1} x\left(n' + \frac{N}{2}\right) W_N^{kn'} W_N^{k\frac{N}{2}}
\end{aligned}
$$

Radix-2 FFT: Decimation-in-frequency

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

Radix-2 FFT: Decimation-in-frequency

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n + \frac{N}{2}\right)\right] W_N^{kn}, \quad k = 0, 1, \ldots, N \\ X(2k) &= \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^{2k} x\left(n + \frac{N}{2}\right)\right] W_N^{2kn} \\ &= \sum_{n=0}^{(N/2)-1} \underbrace{\left[x(n) + x\left(n + \frac{N}{2}\right)\right]}_{\equiv g_1(n)} W_{N/2}^{kn}, \quad k = 0, 1, \ldots, N/2 \\ X(2k+1) &= \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^{2k+1} x\left(n + \frac{N}{2}\right)\right] W_N^{(2k+1)n} \\ &= \sum_{n=0}^{(N/2)-1} \underbrace{\left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n}_{\equiv g_2(n)} W_{N/2}^{kn}, \quad k = 0, 1, \ldots, N/2 \end{aligned}$$

Radix-2 FFT: Decimation-in-frequency

Therefore,

$$\begin{aligned} X(2k) &= \sum_{n=0}^{(N/2)-1} g_1(n) W_{N/2}^{kn} \\ X(2k+1) &= \sum_{n=0}^{(N/2)-1} g_2(n) W_{N/2}^{kn} \end{aligned}$$

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

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: $O(N \log N)$

FFT Algorithms and radix

- 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 $n = 0, 1, \ldots, N-1$.

$$\begin{aligned} y(n) &= x(n) * h(n) \\ &= \sum_{k=-\infty}^{\infty} x(k)h(n-k) \\ &= \sum_{k=0}^{N-1} x(k)h(n-k) \qquad n = 0, 1, \ldots, N-1 \end{aligned}$$

Convolution and Complexity

Complexity of doing a brute-force convolution is given by:

- For fixed n:

$$y(n) = \sum_{k=0}^{N-1} x(k) \bullet h(n-k)$$

  - N real multiplications
  - $N-1$ real additions
- For all n (n=0, 1, ..., 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 $n = 0, 1, \ldots, N-1$.
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 $y(n)$, $n = 0, 1, \ldots, N-1$.

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. ■