Computation of the DFT

Carrson C. Fung

Institute of Electronics

National Yang Ming Chiao Tung University

Overview

- Efficient algorithms for computing the DFT
  - Fast Fourier transform (FFT)
    - Many types of FFTs
      - Radix-2 (DIT, DIF), Composite N (Cooley-Tukey), Winograd, Chirp transform, …
    - Principle lies in divide and conquer
- Efficient criteria
  - Number of multiplications
  - Number of additions
  - Chip area in VLSI implementation

DFT as a Linear Transformation

- Recall that the DFT can be regarded as linear transformation
  - Can be computed using matrices
- DFT:

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

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

- Let

$$\mathbf{x}_N = \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}, \quad \mathbf{X}_N = \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix},$$

and

$$\text{IDFT matrix: } \mathbf{W}_N^{nk} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}, \text{ for } k,n = 0,1\ldots,N-1$$

DFT as a Linear Transformation (cont'd)

- Note that for the matrix notation, I split 1/N in the synthesis equation to both the IDFT and DFT matrix

$$\mathbf{X}_N = \mathbf{W}_N^H \mathbf{x}_N \qquad N\text{-point DFT}$$

$$\mathbf{x}_N = \mathbf{W}_N \mathbf{X}_N \qquad N\text{-point IDFT } (\mathbf{W}_N \text{ is an unitary matrix})$$

$$\text{where } \left[\mathbf{W}_N\right]_{nk} = \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}kn}, \quad n,k = 0,1,\ldots,N-1$$

- Because the matrix (transformation) $\mathbf{W}_N$ has a specific structure and because W_N^k has particular values (for some k and n), we can reduce the number of arithmetic operations for computing this transform

Example

- Let $\mathbf{x} = \begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix}^T$
- Compute the 4-point DFT
- DFT matrix

$$\mathbf{W}_4^H = \frac{1}{2}\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-j\frac{2\pi}{4}\cdot 1} & e^{-j\frac{2\pi}{4}\cdot 2} & e^{-j\frac{2\pi}{4}\cdot 3} \\ 1 & e^{-j\frac{2\pi}{4}\cdot 2} & e^{-j\frac{2\pi}{4}\cdot 4} = W_4^0 & e^{-j\frac{2\pi}{4}\cdot 6} = W_4^2 \\ 1 & e^{-j\frac{2\pi}{4}\cdot 3} = W_4^3 & e^{-j\frac{2\pi}{4}\cdot 6} = W_4^2 & e^{-j\frac{2\pi}{4}\cdot 9} = W_4^1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

- Thus

$$\mathbf{X}_4 = \mathbf{W}_4^H \mathbf{x}_4 = \begin{bmatrix} 3 \\ -1+j \\ -1 \\ -1-j \end{bmatrix}$$

- Note that to compute each $X[k]$ value, assuming $x[n]$ is complex-valued, it requires us to perform N complex-valued multiplications and N-1 complex-valued additions. To compute all N values of the FFT, it requires N^2 complex-valued multiplications and $N(N\text{-}1)$ additions

Fast Fourier Transform

- Highly efficient algorithms for computing DFT
  - General principle: *Divide and conquer*
  - Exploit specific properties of W_N^k
    - Complex conjugate symmetry: $W_N^{kn} = (W_N^{kn})^*$
    - Symmetry: $W_N^{k+N/2} = -W_N^k$
    - Periodicity: $W_N^{k+N} = W_N^k$
    - Particular values of k and n: e.g.radix-4 FFT (no multiplications: e.g. multiplication by 1 and -1)

FFT

- Direct computation of DFT

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

$$= \sum_{n=0}^{N-1} \left\{ \begin{array}{l} \left[\mathrm{Re}(x[n]) \cdot \mathrm{Re}\left(W_N^{kn}\right) - \mathrm{Im}(x[n]) \cdot \mathrm{Im}\left(W_N^{kn}\right) \right] + \\ j\left[\mathrm{Re}(x[n]) \cdot \mathrm{Im}\left(W_N^{kn}\right) + \mathrm{Im}(x[n]) \mathrm{Re}\left(W_N^{kn}\right) \right] \end{array} \right\}$$

- For 1 complex multiplication: $x[n]W_N^{kn}$ requires
  - 4 real-valued multiplications, 2 real-valued additions
- For each complex-valued addition
  - 2 real-valued additions
- So, for each k, since we need N complex multiplications and N-1 complex additions
  - N complex-valued multiplications = $4N$ real-valued multiplications and $2N$ real-valued additions
  - N-1 complex-valued additions = $2(N-1)$ real-valued additions
    - Therefore, $4N$ real-valued multiplications and $2N+2(N-1) = 4N-2$ real-valued additions
- Therefore, to compute all N values of the DFT, we need $4N^2$ real-valued multiplications and $N(4N-2)$ real-valued additions.

Radix-2: Decimation-in-Time Algorithms

- Idea: N-point DFT → $N/2$-point DFT → $N/4$-point DFT
  $N/4$-point DFT
- $N/2$-point DFT → $N/4$-point DFT
  $N/4$-point DFT
- Sequence: $x[0]\ x[1]\ x[2]\ x[3]\ \ldots\ x[n/2]\ \ldots\ x[N-1]$
  - Even index: $x[0]\ \ x[2]\ \ldots\ x[N-2]$
  - Odd index: $x[1]\ x[3]\ \ldots\ x[N-1]$

Radix-2: Decimation-in-Time Algorithms (cont'd)

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

$$= \underbrace{\sum_{n \text{ even}} x[n] W_N^{kn}}_{n=2r} + \underbrace{\sum_{n \text{ odd}} x[n] W_N^{kn}}_{n=2r+1}$$

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

Since $W_N^2 = e^{-2j\left(\frac{2\pi}{N}\right)} = e^{-2j\left(\frac{\pi}{N/2}\right)} = W_{N/2}$

$$X[k] = \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{N/2}^{rk}}_{\frac{N}{2}\text{-point DFT}} + W_N^k \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{N/2}^{rk}}_{\frac{N}{2}\text{-point DFT}}$$

$$= G[k] + W_N^k H[k]$$

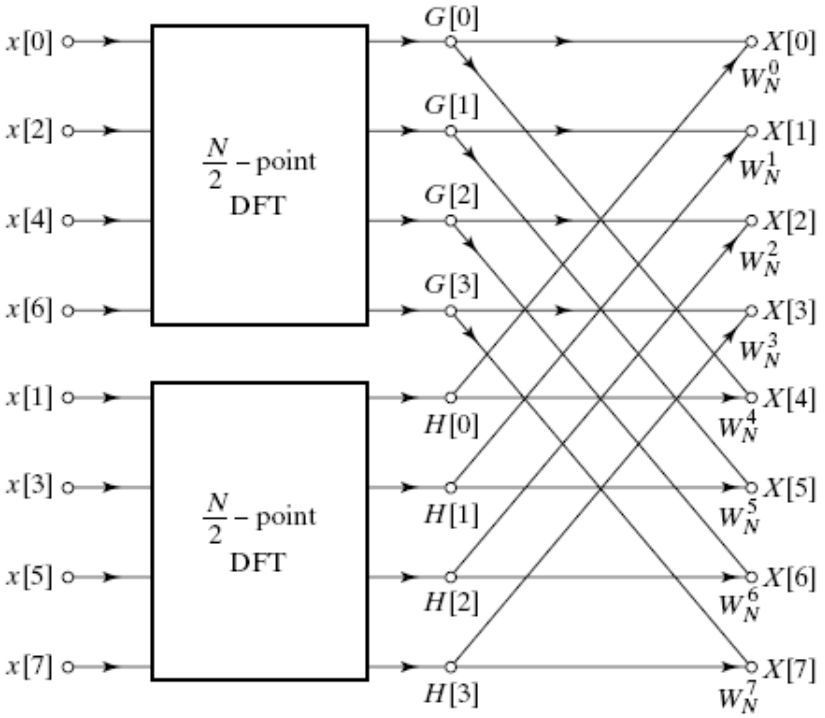


Figure 9.3 Flow graph of the decimation-in-time decomposition of an N-point DFT computation into two $(N/2)$-point DFT computations $(N = 8)$.

Radix-2: Decimation-in-Time Algorithms (cont'd)

- Note that $G[k]$ and $H[k]$ are periodic in k with a period of $N/2$. Therefore for $N = 8$

$$\begin{aligned} X[4] &= G[4] + W_4^4 H[4] \\ &= G[0] + W_4^4 H[0] \end{aligned}$$

- Similar relationship can be exploited to obtain $X[5]$, $X[6]$, and $X[7]$

Comparison

- Direct computation of N-point DFT (N frequency samples)
  - N^2 complex-valued multiplications and ~N^2 (it's actually $N(N\text{-}1)$, but we shall assume N is large) complex-valued additions
- Direct computation of $N/2$-point DFT
  - We need to perform $2(N/2)$-point DFTs, each one will require $(N/2)^2$ complex-valued multiplications and approximately $(N/2)^2$ complex-valued additions. Therefore, we need to perform $2(N/2)^2$ complex-valued multiplications and $2(N/2)^2$ complex-valued additions
  - Combining the $2(N/2)$-point DFTs also requires
    - an additional N complex-valued multiplications (because we need to multiply the W_N^k terms)
    - an additional N complex-valued additions (combining the $G[k]$ and $H[k]$ terms)
  - Adding everything together, we have:
    - $N + 2(N/2)^2 = N + N^2/2$ complex-valued multiplications
    - $N + 2(N/2)^2 = N + N^2/2$ complex-valued additions
  - For $N>2$, $N+N^2/2 < N^2$, therefore, this divide-and-conquer approach does decrease the amount of computations
- $\log_2 N$-stage FFT
  - Since $N=2^v$, we can further break $N/2$-point DFT into two $N/4$-point DFT and so on

$(N/4)$-point DFT: $G[k]$

$$G[k] = \sum_{r=0}^{N/2-1} g[r] W_{N/2}^{rk}$$

$$= \sum_{\ell=0}^{N/4-1} g[2\ell] W_{N/2}^{2k\ell} + \sum_{\ell=0}^{N/4-1} g[2\ell+1] W_{N/2}^{(2\ell+1)k}$$

$$= \sum_{\ell=0}^{N/4-1} g[2\ell] W_{N/4}^{k\ell} + W_{N/2}^{k} \sum_{\ell=0}^{N/4-1} g[2\ell+1] W_{N/4}^{k\ell}$$

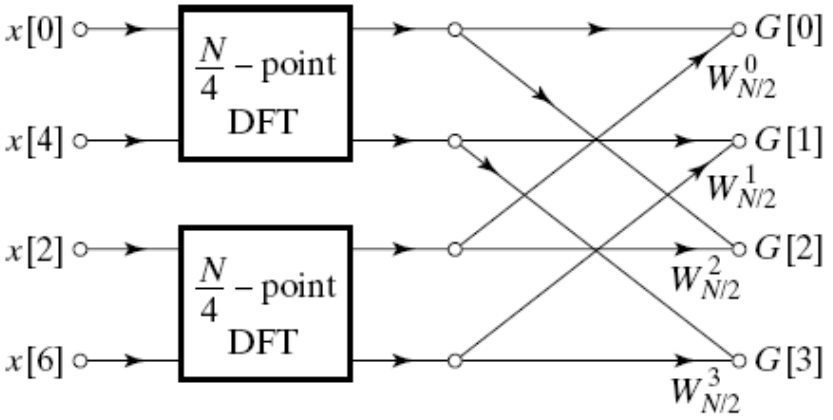


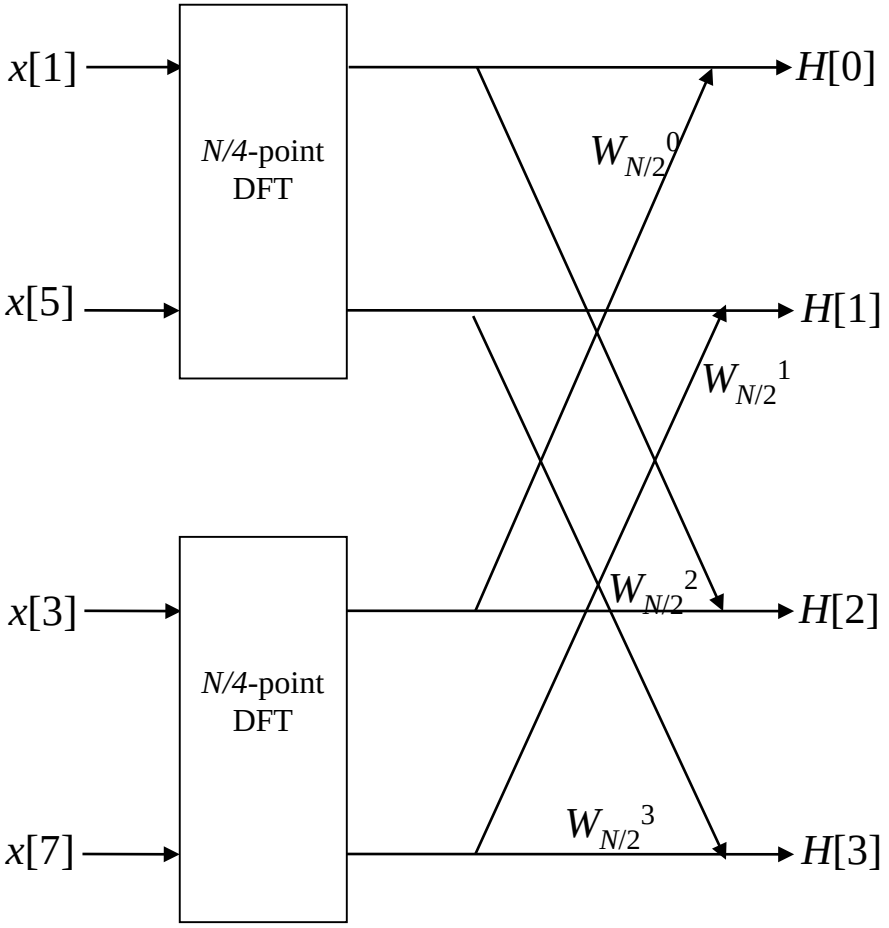
Figure 9.4 Flow graph of the decimation-in-time decomposition of an $(N/2)$-point DFT computation into two $(N/4)$-point DFT computations $(N = 8)$.

$(N/4)$-point DFT: $H[k]$

$$H[k] = \sum_{r=0}^{N/2-1} h[r] W_{N/2}^{rk}$$

$$= \sum_{\ell=0}^{N/4-1} h[2\ell] W_{N/2}^{2k\ell} + \sum_{\ell=0}^{N/4-1} h[2\ell+1] W_{N/2}^{(2\ell+1)k}$$

$$= \sum_{\ell=0}^{N/4-1} h[2\ell] W_{N/4}^{k\ell} + W_{N/2}^{k} \sum_{\ell=0}^{N/4-1} h[2\ell+1] W_{N/4}^{k\ell}$$



Combining $G[k]$ and $H[k]$

- Combining the two figures above and realizing that $W_{N/2}^{k} = W_N^{2k}$, we can obtain Figure 9.5

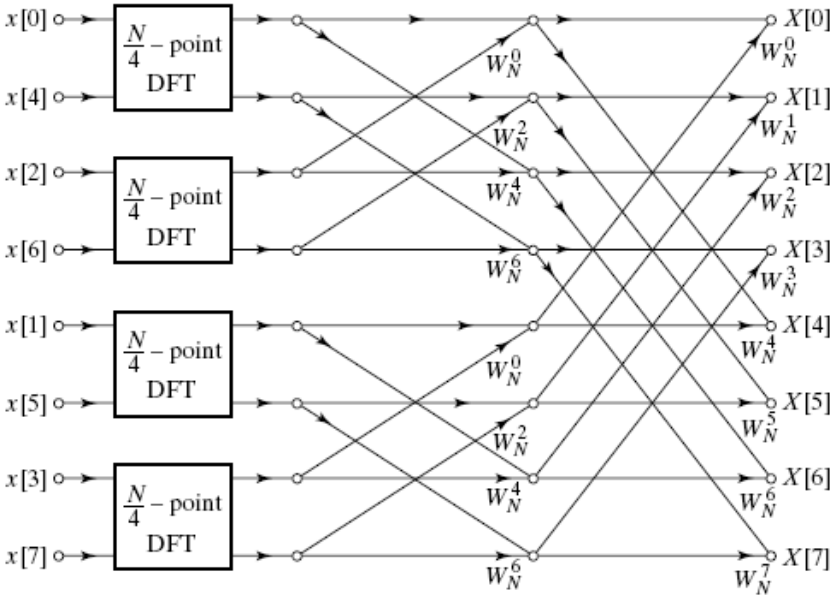


Figure 9.5 Result of substituting the structure of Figure 9.4 into Figure 9.3.

2-point DFT

- Assuming $N = 8$, the $(N/4)$-point DFT is actually a 2-point DFT

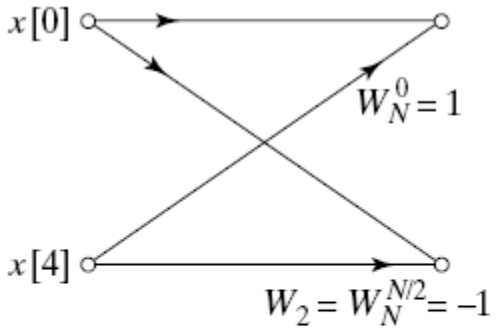


Figure 9.6 Flow graph of a 2-point DFT.

8-point DIT

- Then substituting the 2-point DFT structure in Figure 9.6 into Figure 9.5, we have the structure in Figure 9.7

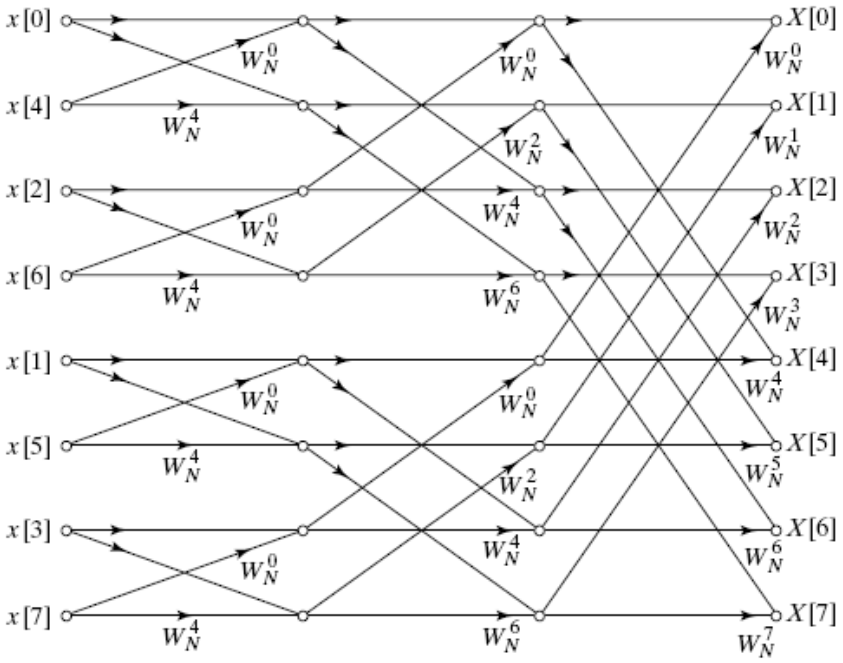


Figure 9.7 Flow graph of complete decimation-in-time decomposition of an 8-point DFT computation.

Number of Operations

- For a general N-point DFT, and assuming N is still a power of 2, we can decompose any N-point DFT into 2 $(N/2)$-point DFT, then the 2 $(N/2)$-point DFTs can be further decomposed into 4 $(N/4)$-point DFTs, then the 4 $(N/4)$-point DFTs can be decomposed into 8 $(N/8)$-point DFTs, and so on until we are left with a 2-point DFT
- There are a total of $v = log_2 N$ number of stages of computations
- Following the same logic as before for the N-point DFT, where we can decompose it into 2 $(N/2)$-point DFT, which requires $N + 2(N/2)^2$ complex-valued multiplications and $N + 2(N/2)^2$ complex-valued additions. Decomposing the $(N/2)$-point DFT into $(N/4)$-point DFT requires us to replace the $(N/2)^2$ term into $N/2 + 2(N/4)^2$. Therefore, we need to perform $N+2(N/2 + 2(N/4)^2) = N + N + 4(N/4)^2$ complex-valued multiplications and $N+2(N/2 + 2(N/4)^2) = N + N + 4(N/4)^2$ complex-valued additions
- This can be done at most v times, so the number of complex-valued multiplications and additions is equal to $Nv = N\log_2 N$

Comparison: DFT vs. FFT

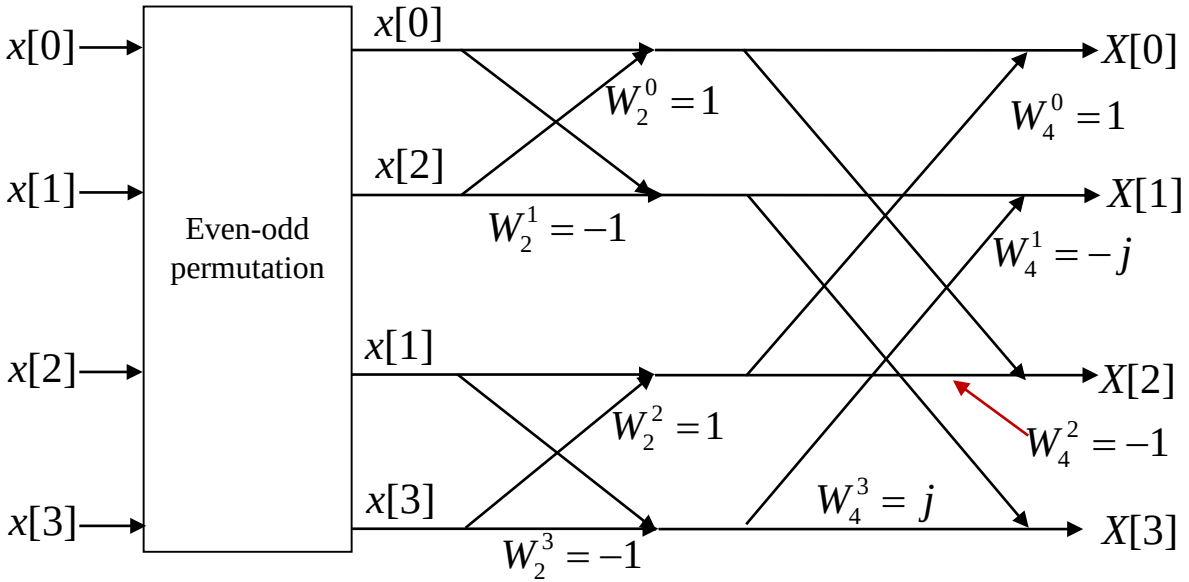
Number of points N	Direct computations: complex-valued multiplications	FFT: Complex-valued multiplications ($N\log_2 N$)	Speed improvement factor
4	16	8	2.0
8	64	24	2.67
16	256	64	4
64	4,096	384	10.67
256	65,536	2,048	32
1024	1,048,576	10,240	102.4

Further savings is possible if we consider the butterfly

FFT in Matrix Form: 4-point FFT

Consider the 4-point DFT matrix and consider decimation-in-time in which the inputs are permuted (bit-reversed) becomes

$$\mathbf{W}_4^H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & & 1 & \\ & 1 & & -j \\ 1 & & -1 & \\ & 1 & & j \end{bmatrix} \begin{bmatrix} 1 & 1 & & \\ 1 & j^2 & & \\ & & 1 & 1 \\ & & 1 & j^2 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix}$$



General 1024-point FFT Decomposition

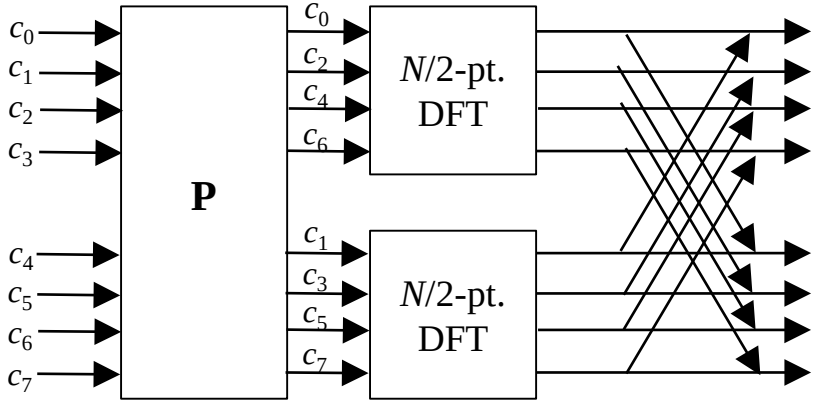
FFT Decomposition

$$\text{For } N=1024, \qquad \mathbf{W}_{1024}^{H}=\begin{bmatrix}\mathbf{I}_{512} & \mathbf{D}_{512}\\ \mathbf{I}_{512} & -\mathbf{D}_{512}\end{bmatrix}\begin{bmatrix}\mathbf{W}_{512}^{H} & \\ & \mathbf{W}_{512}^{H}\end{bmatrix}\begin{bmatrix}\text{even-odd}\\ \text{permutation}\end{bmatrix}$$

$$\mathbf{D}_{512}=\text{Diag}\left(1, e^{-j\frac{2\pi}{N}}, e^{-j\frac{2\pi}{N}2}, \ldots, e^{-j\frac{2\pi}{N}511}\right), \text{ for } N=1024.$$

$$\left[\mathbf{W}_{512}^{H}\right]_{kn}=e^{-j\frac{2\pi}{N}kn}, \text{ for } n,k=0,\ldots,\frac{N}{2}-1$$

E.g. 8-point FFT Decomposition: DIT



N=8 point decimation-in-frequency butterfly

$$\text{For } N=8, \qquad \mathbf{W}_8^H = \begin{bmatrix} \mathbf{I}_4 & \mathbf{D}_4 \\ \mathbf{I}_4 & -\mathbf{D}_4 \end{bmatrix} \begin{bmatrix} \mathbf{W}_4^H & \\ & \mathbf{W}_4^H \end{bmatrix} \begin{bmatrix} \text{even-odd} \\ \text{permutation} \end{bmatrix}$$

$$\mathbf{D}_4 = \text{Diag}\left(1, e^{-j\frac{2\pi}{N}}, e^{-j\frac{2\pi}{N}2}, \ldots, e^{-j\frac{2\pi}{N}7}\right), \quad \text{for } N=8$$

Permutation matrix separates the incoming vector **c** into its even and odd parts

FFT Butterfly

- Butterfly
  - Basic unit in FFT
    - Two multiplications (see Figure 9.8 on right)
- Notice that $W_N^{(r+N/2)} = W_N^{N/2} W_N^r = -1\ W_N^r$, so the butterfly with two multiplications can be turned into a butterfly with a single multiplication (multiplication with 1 and -1 don't count). See Figure 9.9 on right

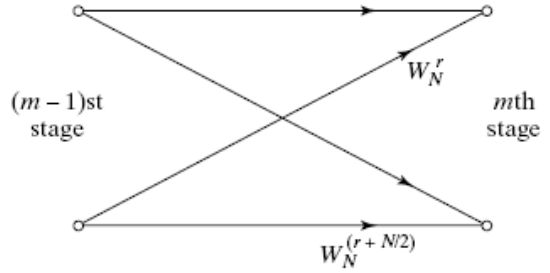


Figure 9.8 Flow graph of basic butterfly computation in Figure 9.7.

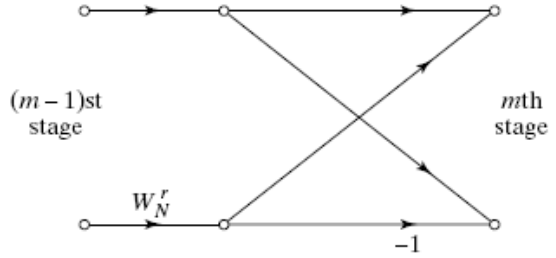


Figure 9.9 Flow graph of simplified butterfly computation requiring only one complex multiplication.

8-point DIT Using Simplified Butterfly Structure

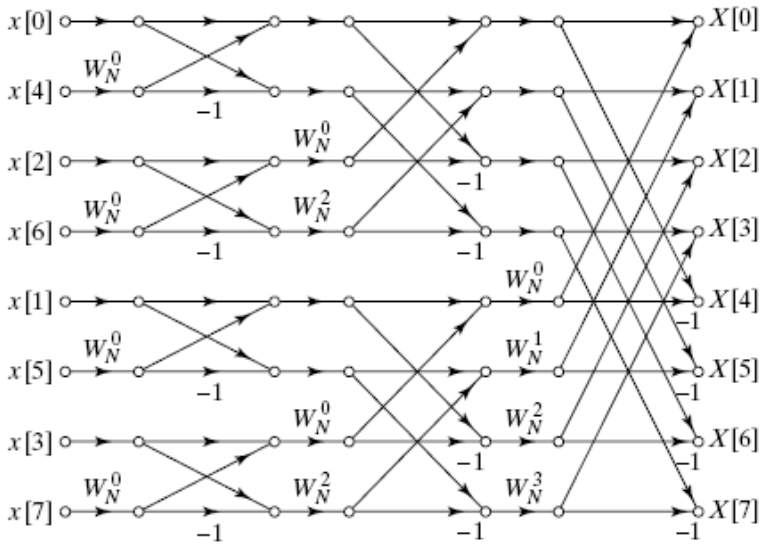


Figure 9.10 Flow graph of 8-point DFT using the butterfly computation of Figure 9.9.

Now we have an additional savings by a factor of *2* since at every butterfly, we have halved the number of multiplications!

In-Place Computations

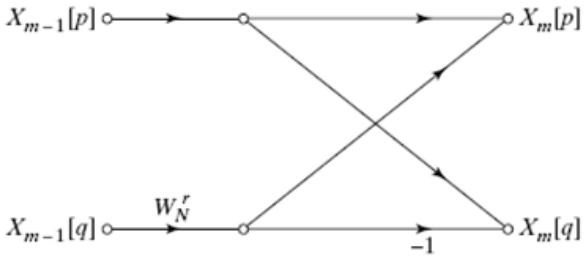


Figure 9.11 Flow graph of Eqs. (9.21).

- Besides efficient computation of the DFT, the structure in Figure 9.10 also offers an efficient way to storage the original data and the results of the computations in the intermediate stages
- Each stage of the FFT process has N inputs and N outputs, so we need exactly N storage locations at any one point in the calculations
- It is possible to reuse the same storage locations at each stage to reduce memory overhead
  - Any algorithm which uses the same memory to store successive iterations of a calculation is called an "in-place" algorithm
  - Computation must be done in a specific order
- Let $X_m[p]$ and $X_m[q]$ denote the results from the m^{th} stage of computations and $X_{m-1}[p]$ and $X_{m-1}[q]$ denote the results from the $(m\text{-}1)^{th}$ stage of computations. Then the relationship between the input and output of the butterfly can be written as (see Figure 9.11)

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q]$$
$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]$$

Only two registers are needed for computing a butterfly unit because $X_m[p]$ and $X_m[q]$ are stored in the same storage registers as $X_{m-1}[p]$ and $X_{m-1}[q]$, respectively. This is because once $X_m[p]$ and $X_m[q]$ are produced from $X_{m-1}[p]$ and $X_{m-1}[q]$, there is no need to store $X_{m-1}[p]$ and $X_{m-1}[q]$ anymore, so we can place the new results, $X_m[p]$ and $X_m[q]$, into the storage location of $X_{m-1}[p]$ and $X_{m-1}[q]$

In-Place Computations (cont'd)

In order to retain the in-place computation property, the input data are accessed in the **bit-reversed** order. This gives us an easy way to index the data

Note: The outputs are in the normal order (same as the "position")

Position	Binary equivalent	Bit reversed	Sequence index
6	110	011	3
2	010	010	2

Remark: Index 3 input data is placed at position 6

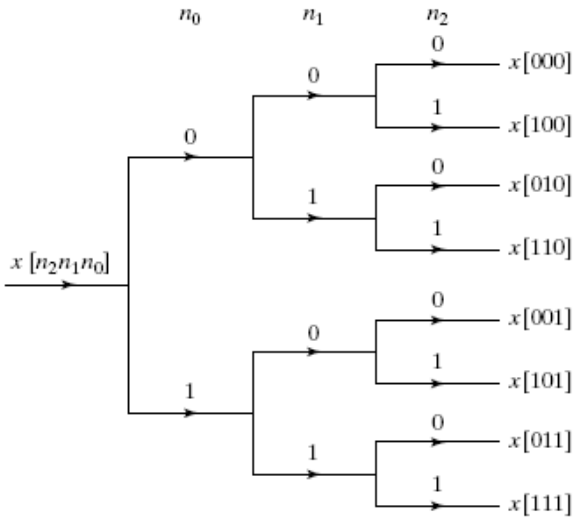


Figure 9.13 Tree diagram depicting bit-reversed sorting.

In-Place Computations (cont'd)

- We may also place the inputs in the normal order; then the outputs are in the bit-reversed order to obtain the structure in Figure 9.14

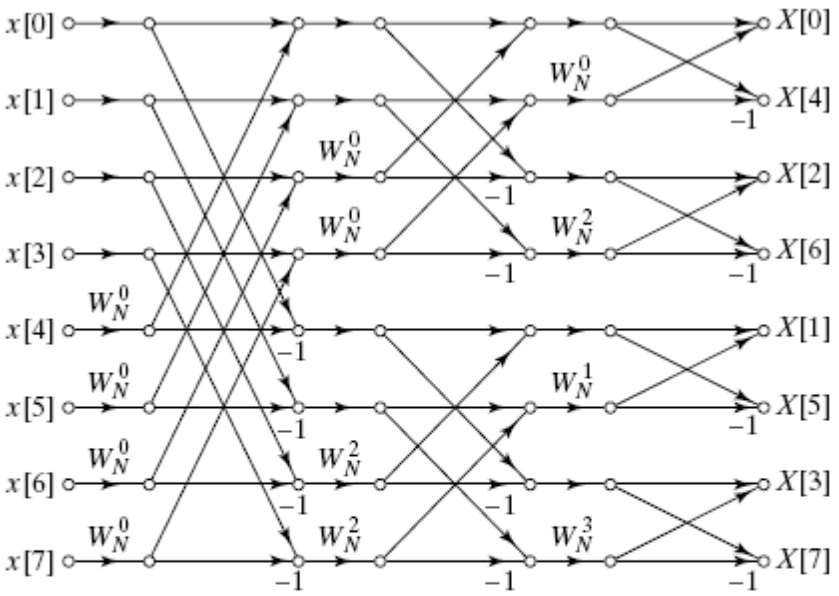


Figure 9.14 Rearrangement of Figure 9.10 with input in normal order and output in bit-reversed order.

In-Place Computations (cont'd)

- If we try to maintain the normal order of both inputs and outputs, then in-place computation structure is destroyed. The structure is shown in Figure 9.15. There is no advantage in using this structure since there is no in-place computation (cannot save storage space), and no easy way to index the data (the input data are not stored in bit-reversed order)

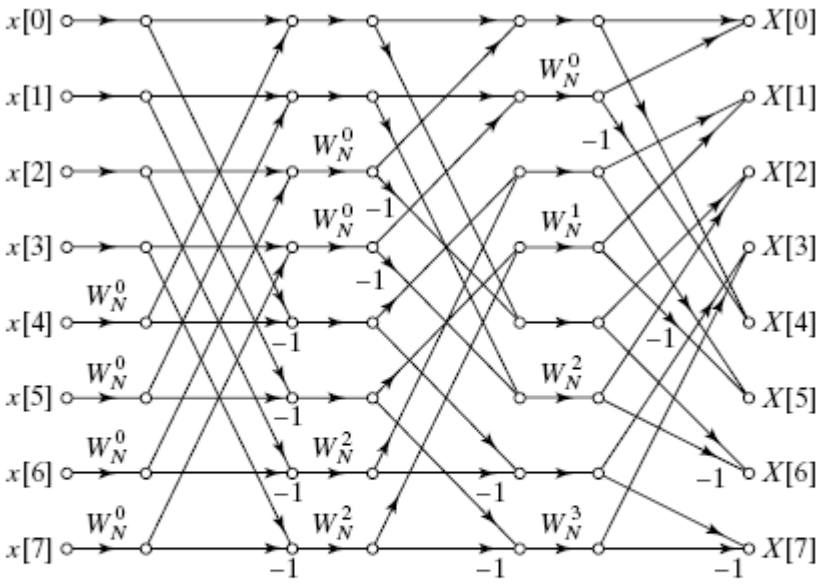


Figure 9.15 Rearrangement of Figure 9.10 with both input and output in normal order.

Radix-2 Decimation-in-Frequency Algorithms

Dividing the output sequence $X[k]$ into smaller pieces

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

Assuming $N = 2^v$, where v is an integer. If k is even, i.e. $k = 2r$, then

$$X[2r]=\sum_{n=0}^{N-1}x[n]W_N^{kn},\qquad r=0,1,\cdots,\frac{N}{2}-1$$

$$=\sum_{n=0}^{\frac{N}{2}-1}x[n]W_N^{2nr}+\sum_{n=\frac{N}{2}}^{N-1}x[n]W_N^{2nr}\qquad n\leftarrow(n+N/2)\text{ in second term}$$

$$=\sum_{n=0}^{\frac{N}{2}-1}x[n]W_N^{2nr}+\sum_{n=0}^{\frac{N}{2}-1}x\left[n+\frac{N}{2}\right]\cdot W_N^{2r\left(n+\frac{N}{2}\right)}$$

$$\because W_N^{2r[n+N/2]}=W_N^{2rn}W_N^{rN}=W_N^{2rn}$$

$$=\sum_{n=0}^{\frac{N}{2}-1}\left(x[n]+x\left[n+\frac{N}{2}\right]\right)\cdot W_N^{2nr}$$

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

Radix-2 Decimation-in-Frequency Algorithms

For $k = 2r+1,\ r = 0,1,\ldots,\frac{N}{2}-1$

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

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

Let $n \to n+\frac{N}{2}$, then $\sum_{n=\frac{N}{2}}^{N-1} x[n] W_N^{n(2r+1)} = \sum_{n=0}^{\frac{N}{2}-1} x\left[n+\frac{N}{2}\right] W_N^{\left[n+\frac{N}{2}\right](2r+1)}$

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

Radix-2 Decimation-in-Frequency Algorithms

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

Note $W_N^{\left(\frac{N}{2}\right)(2r+1)}=W_N^{\left(\frac{N}{2}2r\right)}W_N^{\frac{N}{2}}=1\cdot -1$. Therefore

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

So

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

$$W_N^{2nr}=W_{N/2}^{nr}$$

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

Radix-2 Decimation-in-Frequency Algorithms (cont'd)

$$\begin{cases} X[2r] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n+\frac{N}{2}\right] \right) \cdot W_{N/2}^{nr} \\ X[2r+1] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] - x\left[n+\frac{N}{2}\right] \right) W_N^n W_{N/2}^{nr} \end{cases}$$

$$\begin{cases} g[n] = x[n] + x\left[n+\frac{N}{2}\right] \\ h[n] = x[n] - x\left[n+\frac{N}{2}\right] \end{cases}$$

$W_{N/2}^{nr}$ is refers to the $N/2$-point DFT

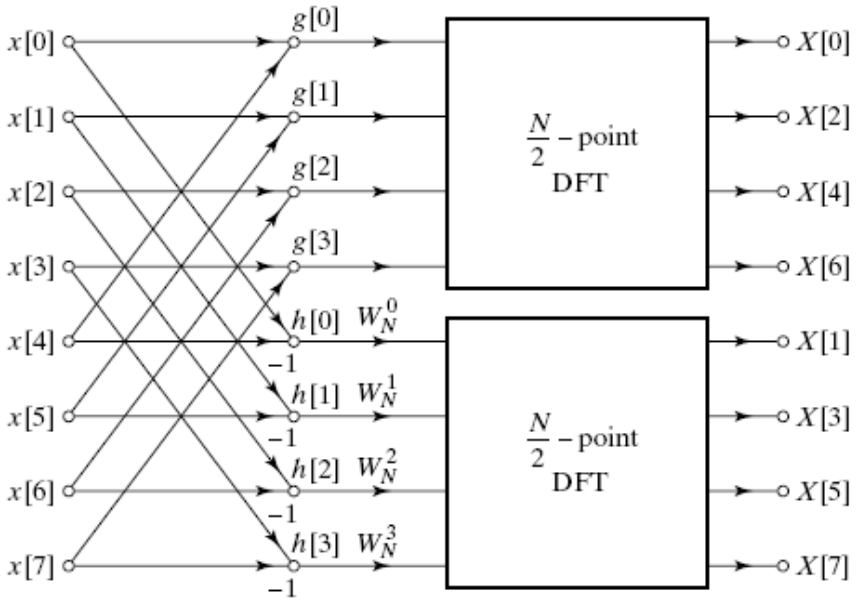


Figure 9.17 Flow graph of decimation-in-frequency decomposition of an N-point DFT computation into two $(N/2)$-point DFT computations $(N = 8)$.

Radix-2 Decimation-in-Frequency Algorithms (cont'd)

- We can further break into even and odd groups (just like DIT). Again, we can reduce the two-multiplication butterfly into one multiplication. Hence, the computational complexity is about $(N/2)\log_2 N$. The in-place computation property holds if the outputs are in bit-reversed order (when inputs are in the normal order)

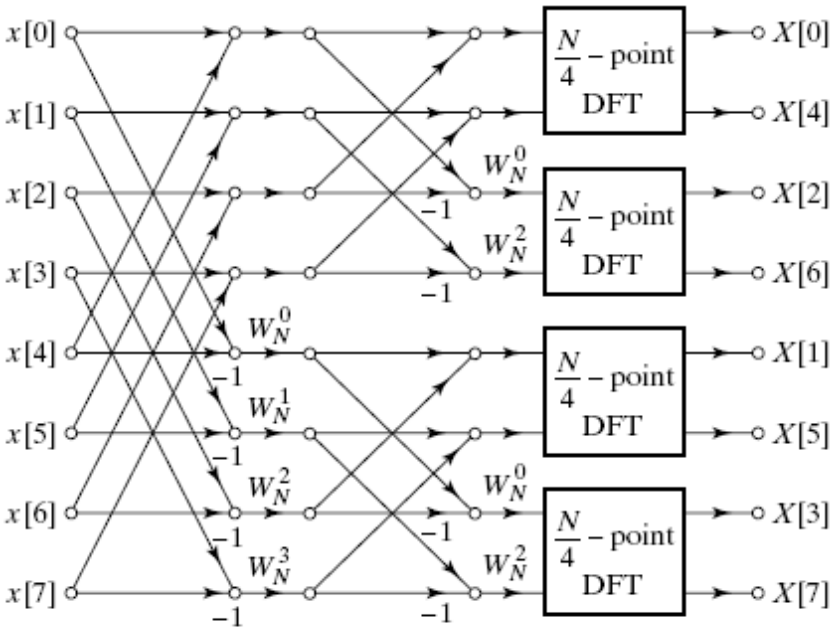


Figure 9.18 Flow graph of decimation-in-frequency decomposition of an 8-point DFT into four 2-point DFT computations.

Radix-2 Decimation-in-Frequency Algorithms (cont'd)

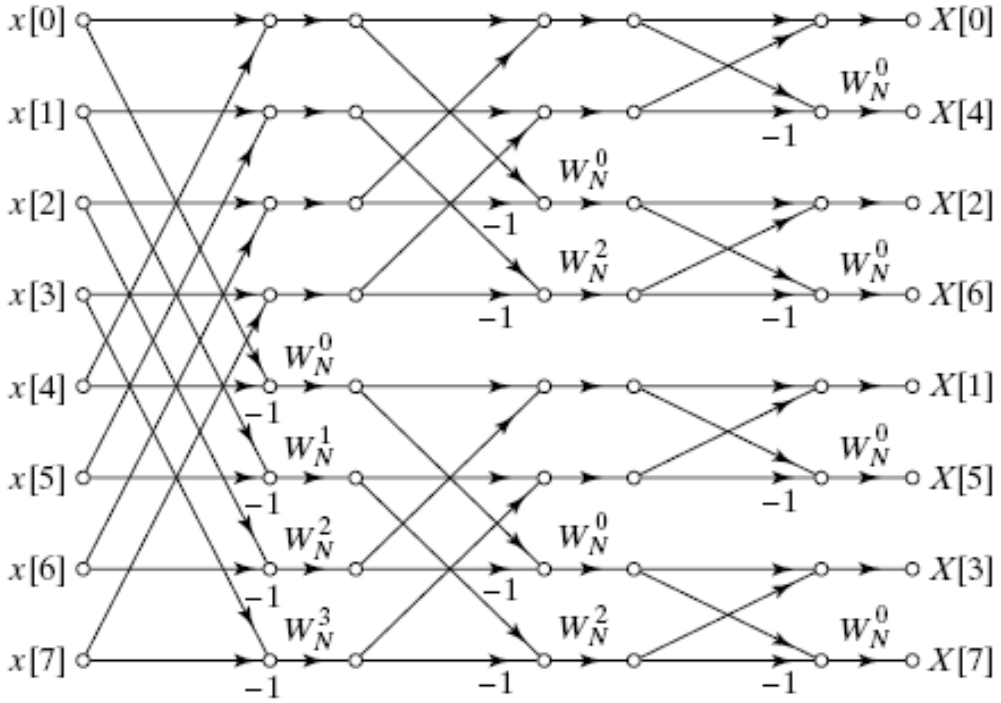


Figure 9.20 Flow graph of complete decimation-in-frequency decomposition of an 8-point DFT computation.

FFT for Composite N

- When N is not a power of 2, but a product of 2 or more integers, FFT algorithms still exist to compute the DFT
- If N is a prime number, we can simply pad zeros to make N into an composite integer
- Cooley-Tukey Algorithm: $N = N_1 N_2$
- DIT and DIF are special cases of Composite N algorithm where $N = 2(N/2) = (N/2)2$
- For composite N FFT, we can divide $x[n]$ and $X[k]$ into a two-dimensional array

Example

$$\begin{cases} \text{Time index:} & n = N_2 n_1 + n_2 \text{ or } n = n_1 + N_1 n_2 & \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \\ \text{Freq. index:} & k = N_2 k_1 + k_2 \text{ or } k = k_1 + N_1 k_2 & \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \end{cases}$$

- For $x[n]$ in this case, n_1 is the column index and n_2 is the row index. There will be N_2 elements in each column (i.e. N_2 number of rows) of the two-dimensional array which stores $x[n]$.
- For $X[k]$, k_1 is the column index and k_2 is the row index. There will be N_1 elements in each row (i.e. N_1 number of columns) of the two-dimensional array which stores $X[k]$

Example (cont'd)

For example: $N_1 = 2$, $N_2 = N/N_1 = N/2$, then the input signal $x[n]$ can be arranged into a 2-dimensional array as

n_2 \ n_1	0	1
0	$x[0]$	$x[N/2]$
1	$x[1]$	$x[N/2+1]$
.	.	.
.	.	.
N_2-1	$x[N/2-1]$	$x[N-1]$

Decomposition

Goal: Decompose N-point DFT into two stages: let $n = N_2 n_1 + n_2$ and $k = k_1 + N_1 k_2$

N_1-point DFT$\otimes N_2$-point DFT

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

$$= X[k_1 + N_1 k_2]$$

$$= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_N^{(k_1 + N_1 k_2)(N_2 n_1 + n_2)}$$

$$= \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot \underbrace{W_N^{N_2 k_1 n_1}}_{W_{N_1}^{k_1 n_1}} \cdot W_N^{k_1 n_2} \cdot \underbrace{W_N^{k_2 N_1 n_2}}_{W_{N_2}^{k_2 n_2}} \cdot \underbrace{W_N^{N_1 N_2 k_2 n_1}}_{1}$$

$$= \underbrace{\sum_{n_2=0}^{N_2-1} \left\{ \underbrace{\left[\sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_{N_1}^{k_1 n_1} \right]}_{N_1\text{-point}} \cdot \underbrace{W_N^{k_1 n_2}}_{\substack{\text{twiddle} \\ \text{factor}}} \right\} \cdot W_{N_2}^{k_2 n_2}}_{N_2\text{-point}}$$

Procedure

(1) Compute N_1-point DFT of all N_2 rows: (row transform)

$$G[n_2, k_1] = \sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2] \cdot W_{N_1}^{k_1 n_1}, \quad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases}$$

(2) Each row DFTs are multiplied by twiddle factors:

$$\tilde{G}[n_2, k_1] = W_N^{k_1 n_2} \cdot G[n_2, k_1], \quad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases}$$

(3) Compute N_2-point DFT: (column transform)

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} \tilde{G}[n_2, k_1] \cdot W_{N_2}^{k_2 n_2}$$

Example: N=15 = 3×5 = $N_1 \times N_2$

First, perform a 3-point DFT for each of the five rows (*Notations*: $x[n_2,n_1]$, $G[n_2,k_1]$):

Row 1	$x[0,0] = x[0]$	$x[0,1] = x[5]$	$x[0,2] = x[10]$	→ $G[0,k_1]$
Row 2	$x[1,0] = x[1]$	$x[1,1] = x[6]$	$x[1,2] = x[11]$	→ $G[1,k_1]$
Row 3	$x[2,0] = x[2]$	$x[2,1] = x[7]$	$x[2,2] = x[12]$	→ $G[2,k_1]$
Row 4	$x[3,0] = x[3]$	$x[3,1] = x[8]$	$x[3,2] = x[13]$	→ $G[3,k_1]$
Row 5	$x[4,0] = x[4]$	$x[4,1] = x[9]$	$x[4,2] = x[14]$	→ $G[4,k_1]$

Example: N=15 = 3×5 = $N_1 \times N_2$

Multiply by $W_N^{k_1 n_2}$

Column 1	Column 2	Column 3
$\tilde{G}[0,0]$	$\tilde{G}[0,1]$	$\tilde{G}[0,2]$
$\tilde{G}[1,0]$	$\tilde{G}[1,1]$	$\tilde{G}[1,2]$
$\tilde{G}[2,0]$	$\tilde{G}[2,1]$	$\tilde{G}[2,2]$
$\tilde{G}[3,0]$	$\tilde{G}[3,1]$	$\tilde{G}[3,2]$
$\tilde{G}[4,0]$	$\tilde{G}[4,1]$	$\tilde{G}[4,2]$

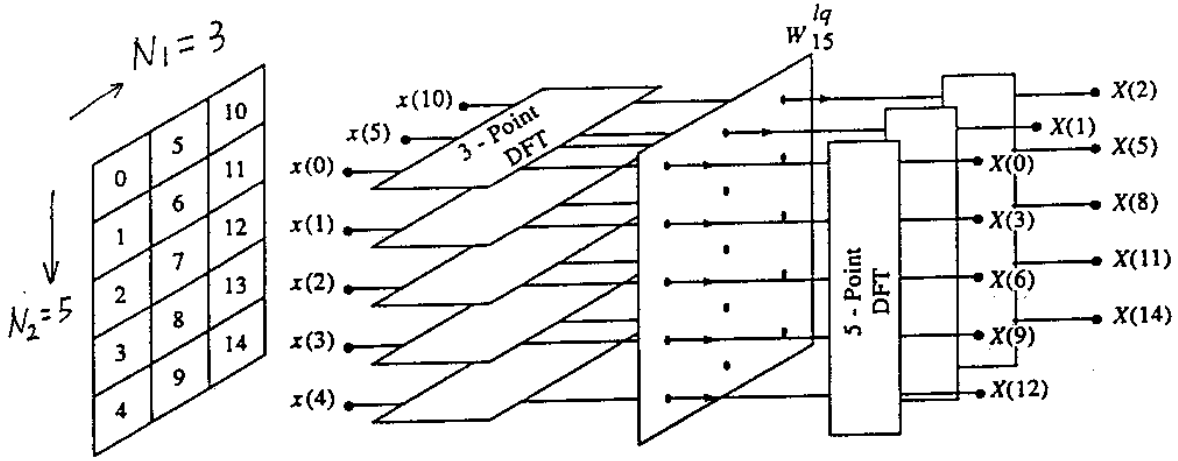
Example: N=15 = 3×5 = $N_1 \times N_2$

Compute the 5-point DFT for each of the three columns

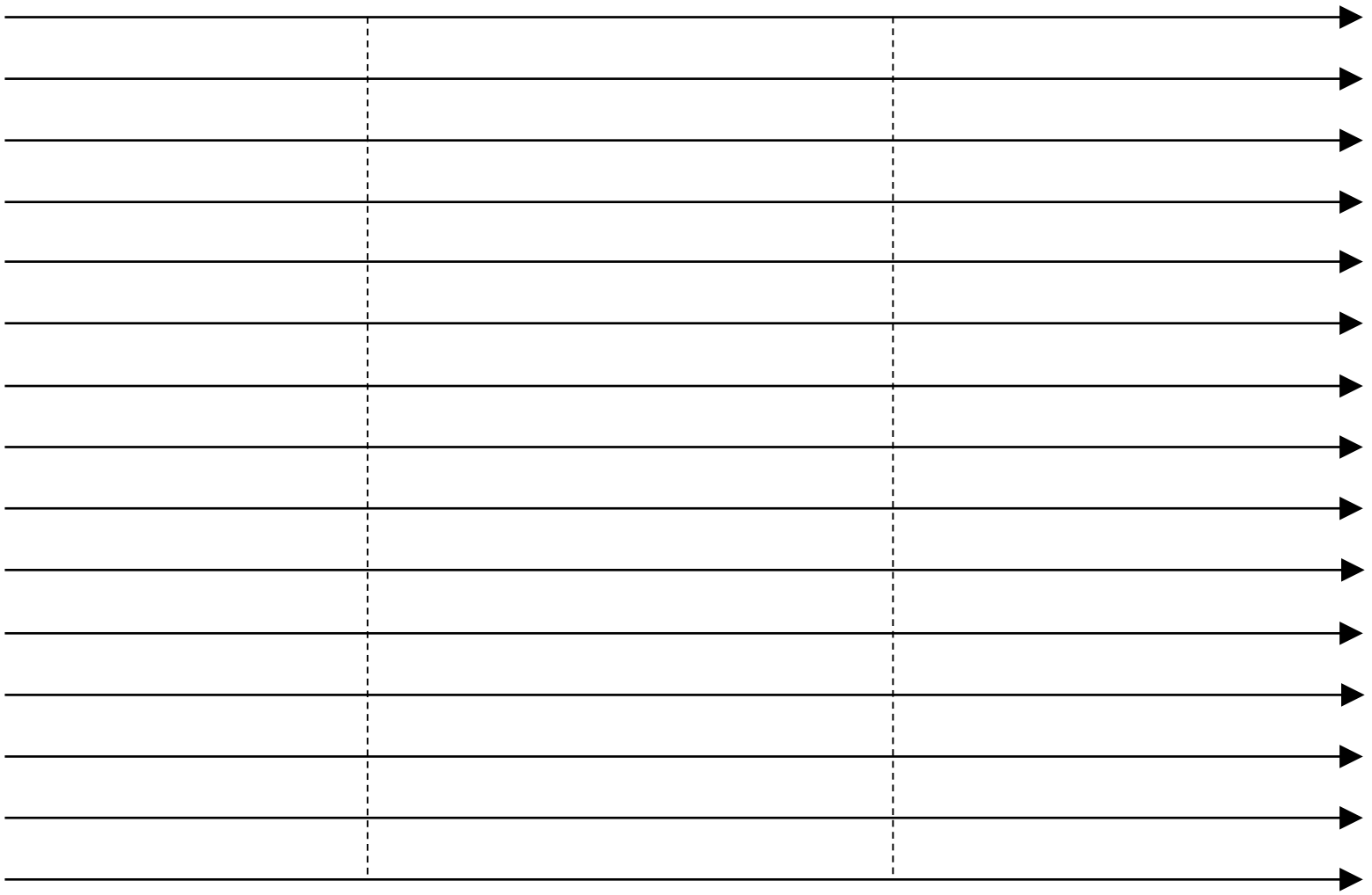
$X[0,0] = X[0]$	$X[0,1] = X[1]$	$X[0,2] = X[2]$
$X[1,0] = X[3]$	$X[1,1] = X[4]$	$X[1,2] = X[5]$
$X[2,0] = X[6]$	$X[2,1] = X[7]$	$X[2,2] = X[8]$
$X[3,0] = X[9]$	$X[3,1] = X[10]$	$X[3,2] = X[11]$
$X[4,0] = X[12]$	$X[4,1] = X[13]$	$X[4,2] = X[14]$

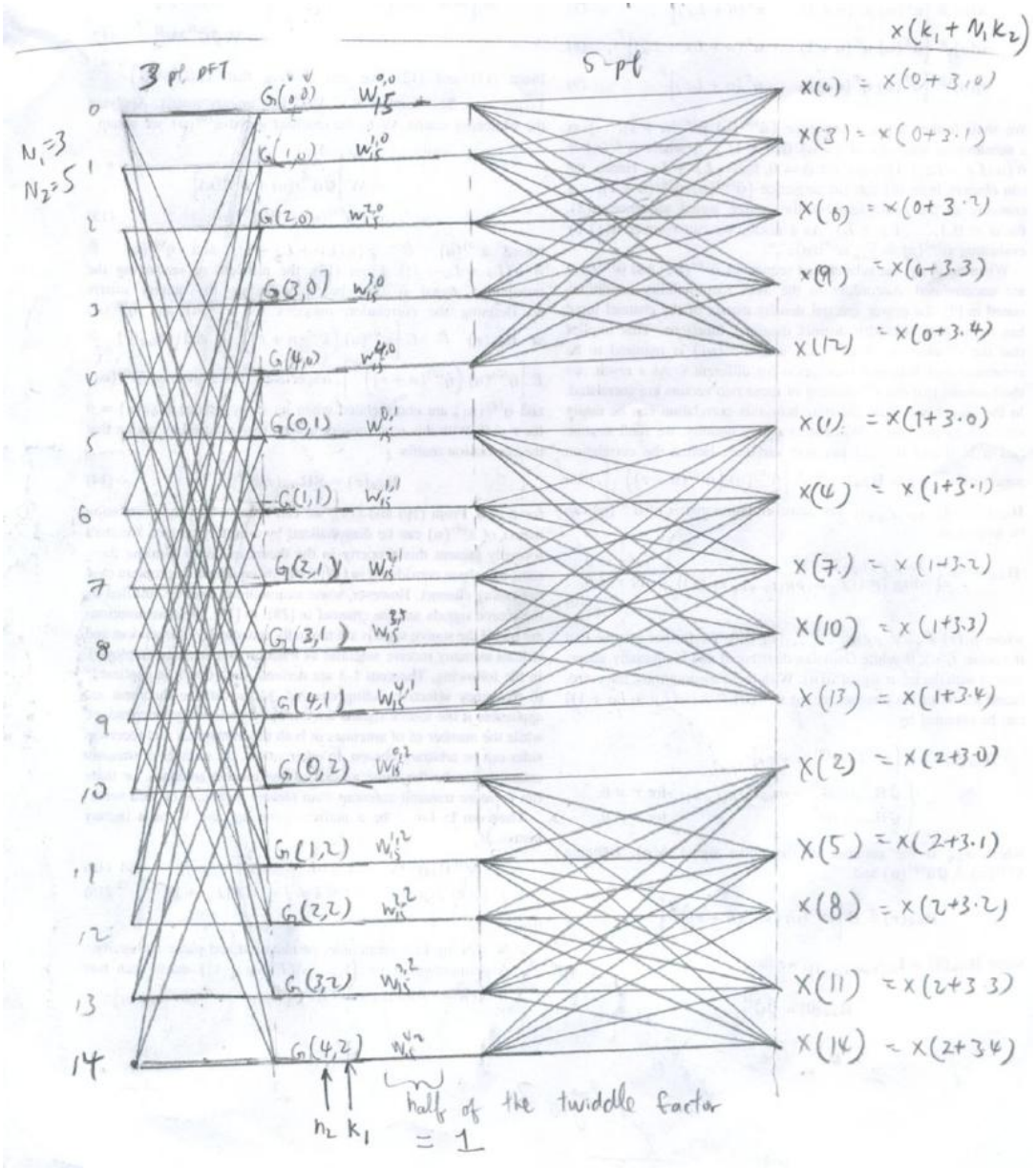
Example: N=15 = 3×5 = $N_1 \times N_2$

Computation of N=15-point DFT by means of 3-point and 5-point DFTs



Example: N=15 = 3×5 = N_1×N_2 - Butterfly Structure




3 pt DFT
5-pt
$N_1=3$
$N_2=5$
$x(k_1+N_1k_2)$
$G(0,0)$ $W_{15}^{0,0}$
$G(1,0)$ $W_{15}^{1,0}$
$G(2,0)$ $W_{15}^{2,0}$
$G(3,0)$ $W_{15}^{3,0}$
$G(4,0)$ $W_{15}^{4,0}$
$G(0,1)$ $W_{15}^{0,1}$
$G(1,1)$ $W_{15}^{1,1}$
$G(2,1)$ $W_{15}^{2,1}$
$G(3,1)$ $W_{15}^{3,1}$
$G(4,1)$ $W_{15}^{4,1}$
$G(0,2)$ $W_{15}^{0,2}$
$G(1,2)$ $W_{15}^{1,2}$
$G(2,2)$ $W_{15}^{2,2}$
$G(3,2)$ $W_{15}^{3,2}$
$G(4,2)$ $W_{15}^{4,2}$
$x(0) = x(0+3\cdot0)$
$x(3) = x(0+3\cdot1)$
$x(6) = x(0+3\cdot2)$
$x(9) = x(0+3\cdot3)$
$x(12) = x(0+3\cdot4)$
$x(1) = x(1+3\cdot0)$
$x(4) = x(1+3\cdot1)$
$x(7) = x(1+3\cdot2)$
$x(10) = x(1+3\cdot3)$
$x(13) = x(1+3\cdot4)$
$x(2) = x(2+3\cdot0)$
$x(5) = x(2+3\cdot1)$
$x(8) = x(2+3\cdot2)$
$x(11) = x(2+3\cdot3)$
$x(14) = x(2+3\cdot4)$
n_2 k_1
half of the twiddle factor
$= 1$

Computational Complexity

- Extension: $N = N_1 N_2 \ldots N_v$
- If $N = N_1 N_2$
  - 1. row transform: $N_2 \cdot \mu(N_1)$
  - 2. twiddle factors: $N_1 N_2 = N$
  - 3. column transform: $N_1 \cdot \mu(N_2)$
- In the example above, when $N_2 = 5$ and $N_1 = 3$. We first have to perform 5 different 3-point DFT (row transform), so the number of multiplications will be 5 times $\mu(3)$, where $\mu(3)$ is the number of multiplications we need to perform for a 3-point DFT. There are 15 twiddle factor multiplications (counting all trivial multiplications as well). Finally, there are three 5-point DFT to perform in the last stage, therefore, 3 times $\mu(5)$ number of multiplications

$$\mu(N) = N_2 \cdot \mu(N_1) + N_1 \cdot \mu(N_2) + N$$
$$= N\left(\frac{\mu(N_1)}{N_1} + \frac{\mu(N_2)}{N_2} + 1\right) \qquad (*)$$

- In general if $N = N_1 N_2 \ldots N_v$, then by repeatly applying (*), we have $\mu(N) = N\left(\sum_{i=1}^{v} \frac{\mu(N_i)}{N_i} + (v-1)\right)$. This is achieved by continuously breaking down the transform into successively smaller transforms. $N(v\text{-}1)$ accounts for the total number of twiddle factor multiplication. However, it should be $N(v\text{-}1)/2$ because half of the twiddle factors are actually equal to "1" (see example when $N = N_1 N_2 = 3*5$)
- Note that the way we reduce the number of computations is to obtain efficient algorithms for implementing the smaller N_i-point transforms with few than N_i^2 multiplications

Special Case: $N_1 = N_2 = \ldots = N_v = 2$

Radix-2: $N_1 = N_2 = \cdots = N_v = 2$ and $v = \log_2 N$

$\mu(N) = \dfrac{N(v-1)}{2}$ multiplications because $\mu(2)$ requires no multiplications

Radix-4: $N_1 = N_2 = \cdots = N_v = 4$ and $v = \log_4 N$

$\mu(N) = \dfrac{N(v-1)}{2}$ multiplications because $\mu(4)$ requires no multiplications

(multiplications by j and $-j$ can be achieved by interchanging the real and imaginary parts). This FFT has few stages than Radix-2 $\Rightarrow$ fewer mults

$$\mathbf{W}_4 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^0 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

For example, let $N = 16 = N_1 N_2 = 4*4$

$$G[n_2, k_1] = x[n_2] + (-j)^{k_1} x[(N/4) + n_2] + (-1)^{k_1} x[(N/2) + n_2] + (j)^{k_1} x[(3N/4) + n_2],$$

$$\text{for } k_1 = 0,1,2,3; \quad n_2 = 0,1,\ldots (N/4) - 1$$

Radix-4 FFT

Each section

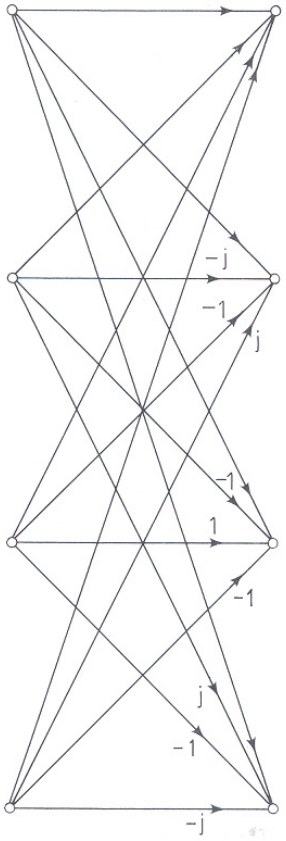


Figure 9.28 Butterfly for 4-point DFT.

Entire structure

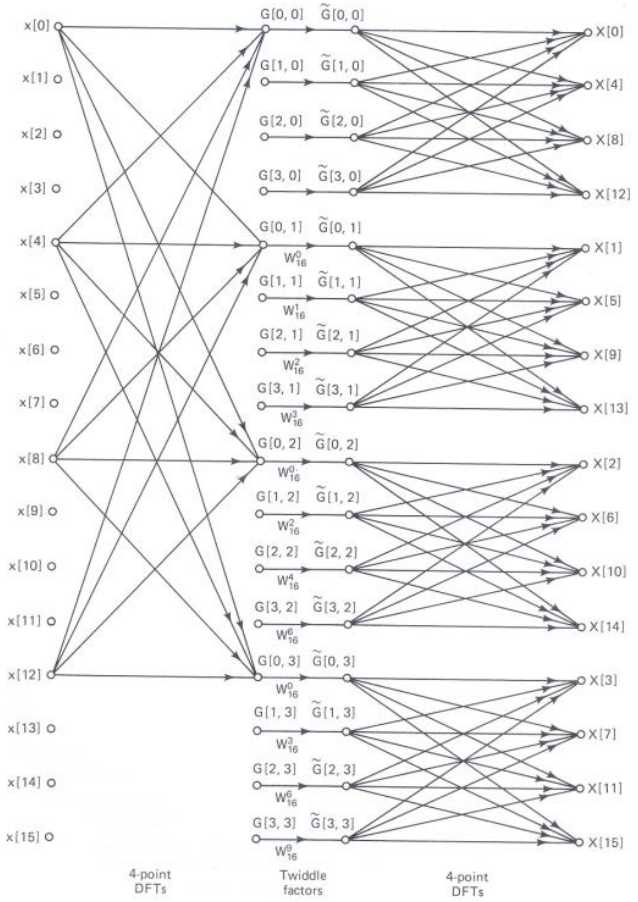


Figure 9.29 Flow graph of $N = 16$ DFT using factors of 4.

Implementation of Inverse FFT Using FFT

IDFT: $x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\cdot W_N^{-kn}$ (*)

DFT: $X[k] = \sum_{n=0}^{N-1} x[n]\cdot W_N^{nk}$

Hence, take the conjugate of (*)

$$x^*[n] = \frac{1}{N}\left(\sum_{k=0}^{N-1} X[k]\cdot W_N^{-kn}\right)^*$$

$$= \frac{1}{N}\sum_{k=0}^{N-1}\left(X^*[k]\cdot W_N^{kn}\right)$$

$$= \frac{1}{N}\,\mathrm{DFT}\left[X^*[k]\right]$$

Then take the conjugate of $x^*[n]$

$$x[n] = \frac{1}{N}\left(\mathrm{DFT}\left[X^*[k]\right]\right)^*$$

$$= \frac{1}{N}\left(\mathrm{FFT}\left[X^*[k]\right]\right)^*$$

Thus, we can use the FFT algorithm to compute the inverse DFT