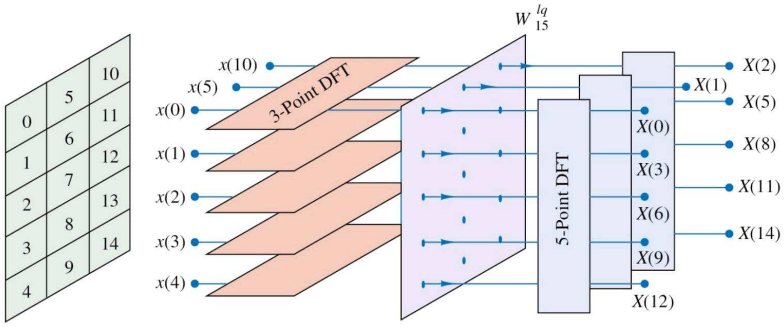
Computation of DFT: Fast Fourier Transform Algorithms



0

Direct DFT Computation

Computation of DFT with complex arithmetic

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

Computation of DFT with real arithmetic

$$X_R(k) = \sum_{n=0}^{N-1} \left[x_R(n) \cos\frac{2\pi}{N} kn + x_I(n) \sin\frac{2\pi}{N} kn \right]$$

$$X_I(k) = -\sum_{n=0}^{N-1} \left[x_R(n) \sin\frac{2\pi}{N} kn - x_I(n) \cos\frac{2\pi}{N} kn \right]$$

Computational operations

- Twiddle factors are pre-computed and stored in look-up tables
- N^2 complex multiplications and $N(N-1) \approx N^2$ complex additions
- Computational complexity: $O(N^2)$
- Indexing and addressing operations

1

MATLAB: Direct Computation

```
function Xdft=dftdirect(x)
% Direct computation of the DFT
N=length(x); Q=2*pi/N;
for k=1:N
      S=0;
      for n=1:N
           W(k,n)=exp(-j*Q*(k-1)*(n-1));
           S=S+W(k,n)*x(n);
      end
      Xdft(k)=S;
end
```


2

Efficient Computation of DFT

- Compute all DFT coefficients as a "block"
  - Fast Fourier Transform (FFT) Algorithms
  - DFT computation algorithms using convolution (WFTA)
- Compute *a* few DFT coefficients or any desired set of samples of DTFT (see Section 8.3 in the Textbook)
  - Goertzel Algorithm
  - Chirp Transform Algorithms (CTA)
- Specialized Algorithms:
  - Sliding DFT (SDFT)
  - The Quick Fourier Transform (QFT)
  - The "Fastest Fourier Transform in the West", see http://www.fftw.org and http://www.mathworks.com (Cleve's Corner, Winter 2001)

3

Twiddle Factor Symmetries

Efficient algorithms exploit the following properties:

1. $W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$ (Periodicity in n and k)
2. $W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^*$ (Complex conjugate symmetry)

$$\boldsymbol{W}_8 = \begin{bmatrix} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \uparrow & \nearrow & \rightarrow & \searrow & \downarrow & \swarrow & \leftarrow & \nwarrow \\ \uparrow & \rightarrow & \downarrow & \leftarrow & \uparrow & \rightarrow & \downarrow & \leftarrow \\ \uparrow & \searrow & \leftarrow & \nearrow & \downarrow & \nwarrow & \rightarrow & \swarrow \\ \uparrow & \downarrow & \uparrow & \downarrow & \uparrow & \downarrow & \uparrow & \downarrow \\ \uparrow & \swarrow & \rightarrow & \nwarrow & \downarrow & \nearrow & \leftarrow & \searrow \\ \uparrow & \leftarrow & \downarrow & \rightarrow & \uparrow & \leftarrow & \downarrow & \rightarrow \\ \uparrow & \nwarrow & \leftarrow & \swarrow & \downarrow & \searrow & \rightarrow & \nearrow \end{bmatrix}$$

$$\uparrow \equiv W_8^0 \quad \nearrow \equiv W_8^1 \quad \rightarrow \equiv W_8^2 \quad \searrow \equiv W_8^3 \quad \downarrow \equiv W_8^4 \quad \swarrow \equiv W_8^5 \quad \leftarrow \equiv W_8^6 \quad \nwarrow \equiv W_8^7$$

4

Concepts through an Example

Example: Let us compute a 4-point DFT and develop an efficient algorithm for this computation.

$$X[k] = \sum_{n-0}^{3} x[n] W_4^{nk}, \quad 0 \le k \le 3; \quad W_4 = \mathrm{e}^{\mathrm{j}2\pi/4} = -j$$

Solution: The above computation can be done in the matrix form

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

which requires 16 complex multiplications.

Efficient Approach: Using periodicity

$$W_4^0 = W_4^4 = 1; \qquad W_4^1 = W_4^9 = -\mathrm{j};$$
$$W_4^2 = W_4^6 = -1; \qquad W_4^3 = \mathrm{j}.$$

5

Concepts through an Example (2)

Efficient Approach: Using periodicity

$$W_4^0 = W_4^4 = 1; \qquad W_4^1 = W_4^9 = -\mathrm{j};$$
$$W_4^2 = W_4^6 = -1; \qquad W_4^3 = \mathrm{j}.$$

and substituting in the above matrix form, we get

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{j} & -1 & \mathrm{j} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{j} & -1 & -\mathrm{j} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

or

$$X[0] = x[0] + x[1] + x[2] + x[3]$$
$$X[1] = x[0] - \mathrm{j}x[1] - x[2] + \mathrm{j}x[3]$$
$$X[2] = x[0] - x[1] + x[2] - x[3]$$
$$X[3] = x[0] + \mathrm{j}x[1] - x[2] - \mathrm{j}x[3]$$

6

Concepts through an Example (3)

Using symmetry we obtain

$$X[0] = x[0] + x[1] + x[2] + x[3]$$
$$= \underbrace{\left(x[0] + x[2]\right)}_{g1} + \underbrace{\left(x[1] + x[3]\right)}_{g2}$$
$$X[1] = x[0] - \mathrm{j}x[1] - x[2] + \mathrm{j}x[3]$$
$$= \underbrace{\left(x[0] - x[2]\right)}_{h1} - \mathrm{j}\underbrace{\left(x[1] - x[3]\right)}_{h2}$$
$$X[2] = x[0] - x[1] + x[2] - x[3]$$
$$= \underbrace{\left(x[0] + x[2]\right)}_{g1} - \mathrm{j}\underbrace{\left(x[1] + x[3]\right)}_{g2}$$
$$X[3] = x[0] + \mathrm{j}x[1] - x[2] - \mathrm{j}x[3]$$
$$= \underbrace{\left(x[0] - x[2]\right)}_{h1} + \mathrm{j}\underbrace{\left(x[1] - x[3]\right)}_{h2}$$

7

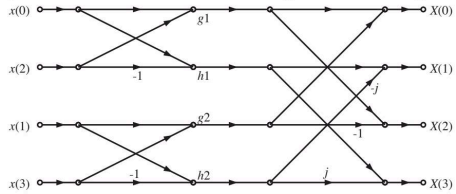
Concepts through an Example (4)

Hence an efficient algorithm is

Step 1			Step 2		
g_1	=	$x[0]+x[2]$	$X[0]$	=	g_1+g_2
g_2	=	$x[1]+x[3]$	$X[1]$	=	h_1-jh_2
h_1	=	$x[0]-x[2]$	$X[2]$	=	g_1-g_2
h_2	=	$x[1]-x[3]$	$X[3]$	=	h_1+jh_2

which requires only 2 complex multiplications, which is a considerably smaller number, even for this simple example.

A signal flowgraph structure for this algorithm is given below:



8

Decimation of Input Sequence

Consider the 8-point DFT given below.

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \\ X[4] \\ X[5] \\ X[6] \\ X[7] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ 1 & W_8^2 & W_8^4 & W_8^6 & 1 & W_8^2 & W_8^4 & W_8^6 \\ 1 & W_8^3 & W_8^6 & W_8 & W_8^4 & W_8^7 & W_8^2 & W_8^5 \\ 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 \\ 1 & W_8^5 & W_8^2 & W_8^7 & W_8^4 & W_8 & W_8^6 & W_8^3 \\ 1 & W_8^6 & W_8^4 & W_8^2 & 1 & W_8^6 & W_8^4 & W_8^2 \\ 1 & W_8^7 & W_8^6 & W_8^5 & W_8^4 & W_8^3 & W_8^2 & W_8 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ x[6] \\ x[7] \end{bmatrix}$$

The even-indexed input and the corresponding DFT matrix row are highlighted in blue.

9

Shuffling of Odd and Even Indexed Samples

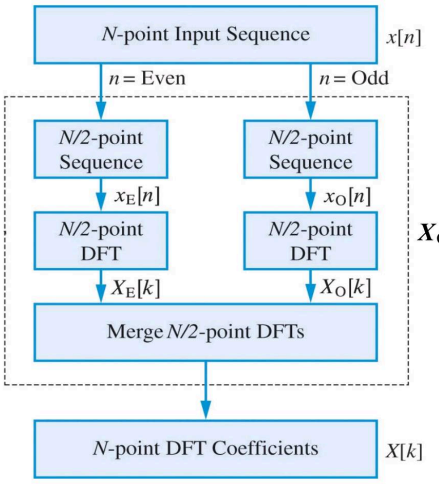
$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{bmatrix} \text{ Top} \qquad \begin{bmatrix} X[4] \\ X[5] \\ X[6] \\ X[7] \end{bmatrix} \text{ Bottom} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8^2 & W_8^4 & W_8^6 & W_8 & W_8^3 & W_8^5 & W_8^7 \\ 1 & W_8^4 & 1 & W_8^4 & W_8^2 & W_8^6 & W_8^2 & W_8^6 \\ 1 & W_8^6 & W_8^4 & W_8^2 & W_8^3 & W_8 & W_8^7 & W_8^5 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & W_8^2 & W_8^4 & W_8^6 & -W_8 & -W_8^3 & -W_8^5 & -W_8^7 \\ 1 & W_8^4 & 1 & W_8^4 & -W_8^2 & -W_8^6 & -W_8^2 & -W_8^6 \\ 1 & W_8^6 & W_8^4 & W_8^2 & -W_8^3 & -W_8 & -W_8^7 & -W_8^5 \end{bmatrix} \begin{bmatrix} x[0] \\ x[2] \\ x[4] \\ x[6] \\ x[1] \\ x[3] \\ x[5] \\ x[7] \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \text{Even} \\ \\ \\ \\ \\ \\ \\ \\ \text{Odd} \end{matrix}$$

$$\boldsymbol{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & 1 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4 \end{bmatrix} \quad \boldsymbol{D}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & W_8 & 0 & 0 \\ 0 & 0 & W_8^2 & 0 \\ 0 & 0 & 0 & W_8^3 \end{bmatrix} \quad \boldsymbol{D}_8\boldsymbol{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ W_8 & W_8^3 & W_8^5 & W_8^7 \\ W_8^2 & W_8^6 & W_8^2 & W_8^6 \\ W_8^3 & W_8 & W_8^7 & W_8^5 \end{bmatrix}$$

$$W_8^2 = W_4$$

10

Decimation-in-Time FFT Algorithm



$$\boldsymbol{X}_E = \boldsymbol{W}_{\frac{N}{2}}\boldsymbol{x}_E \qquad \boldsymbol{X}_O = \boldsymbol{W}_{\frac{N}{2}}\boldsymbol{x}_O$$

$$\begin{aligned} \boldsymbol{X}_T &= \boldsymbol{X}_E + \boldsymbol{D}_N\boldsymbol{X}_O \\ \boldsymbol{X}_B &= \boldsymbol{X}_E - \boldsymbol{D}_N\boldsymbol{X}_O \end{aligned}$$

11

MATLAB: Recursive Computation

```
function Xdft = fftrecur(x)
% Recursive computation of the DFT using divide & conquer
% N should be a power of 2
%
N = length(x);
if N ==1
  Xdft = x;
else
     m = N/2;
     XE = fftrecur(x(1:2:N));
     XO = fftrecur(x(2:2:N));
     W = exp(-2*pi*sqrt(-1)/N).^(0:m-1)';
     temp = W.*XO;
     Xdft = [ XE+temp ; XO-temp ];
end
```


12

Decimation of Output Sequence

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \\ X[4] \\ X[5] \\ X[6] \\ X[7] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ 1 & W_8^2 & W_8^4 & W_8^6 & 1 & W_8^2 & W_8^4 & W_8^6 \\ 1 & W_8^3 & W_8^6 & W_8 & W_8^4 & W_8^7 & W_8^2 & W_8^5 \\ 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 \\ 1 & W_8^5 & W_8^2 & W_8^7 & W_8^4 & W_8 & W_8^6 & W_8^3 \\ 1 & W_8^6 & W_8^4 & W_8^2 & 1 & W_8^6 & W_8^4 & W_8^2 \\ 1 & W_8^7 & W_8^6 & W_8^5 & W_8^4 & W_8^3 & W_8^2 & W_8 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ x[4] \\ x[5] \\ x[6] \\ x[7] \end{bmatrix}$$

13

Decimation-in-Frequency FFT Algorithm

$$\begin{bmatrix} X[0] \\ X[2] \\ X[4] \\ X[6] \\ \hline X[1] \\ X[3] \\ X[5] \\ X[7] \end{bmatrix} \begin{matrix} \text{Even} \\ \\ \\ \\ \\ \text{Odd} \\ \\ \\ \end{matrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8^2 & W_8^4 & W_8^6 & 1 & W_8^2 & W_8^4 & W_8^6 \\ 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 & 1 & W_8^4 \\ 1 & W_8^6 & W_8^4 & W_8^2 & 1 & W_8^6 & W_8^4 & W_8^2 \\ \hline 1 & W_8 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ 1 & W_8^3 & W_8^6 & W_8 & W_8^4 & W_8^7 & W_8^2 & W_8^5 \\ 1 & W_8^5 & W_8^2 & W_8^7 & W_8^4 & W_8 & W_8^6 & W_8^3 \\ 1 & W_8^7 & W_8^6 & W_8^5 & W_8^4 & W_8^3 & W_8^2 & W_8 \end{array}\right] \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \\ \hline x[4] \\ x[5] \\ x[6] \\ x[7] \end{bmatrix} \begin{matrix} \\ \\ \\ \text{Top} \\ \\ \\ \\ \text{Bottom} \end{matrix}$$

$$\begin{bmatrix} \boldsymbol{X}_{\mathrm{E}} \\ \boldsymbol{X}_{\mathrm{O}} \end{bmatrix} = \left[\begin{array}{c|c} \boldsymbol{W}_4 & \boldsymbol{W}_4 \\ \hline \boldsymbol{W}_4\boldsymbol{D}_8 & -\boldsymbol{W}_4\boldsymbol{D}_8 \end{array}\right] \begin{bmatrix} \boldsymbol{x}_{\mathrm{T}} \\ \boldsymbol{x}_{\mathrm{B}} \end{bmatrix}$$

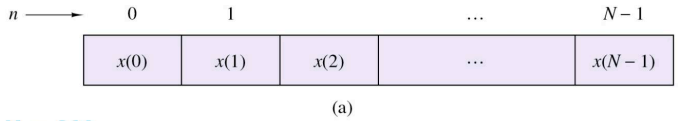
$$\boldsymbol{v} \triangleq \boldsymbol{x}_{\mathrm{T}} + \boldsymbol{x}_{\mathrm{B}} \qquad \boldsymbol{X}_{\mathrm{E}} = \boldsymbol{W}_{\frac{N}{2}}\,\boldsymbol{v}$$

$$\boldsymbol{z} \triangleq \boldsymbol{D}(\boldsymbol{x}_{\mathrm{T}} - \boldsymbol{x}_{\mathrm{B}}) \qquad \boldsymbol{X}_{\mathrm{O}} = \boldsymbol{W}_{\frac{N}{2}}\,\boldsymbol{z}$$

14

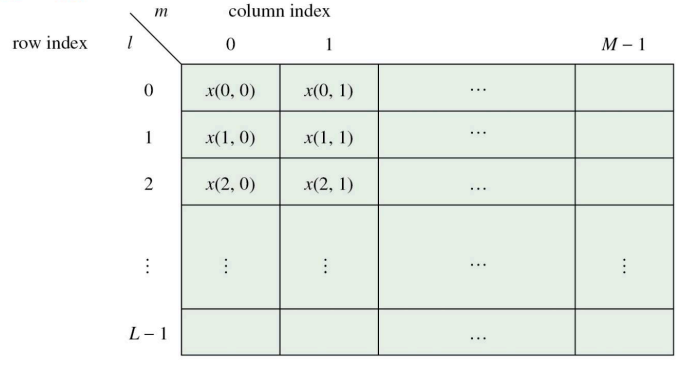
The Divide-and-Conquer Approach

- Step 1 Divide the problem into two or more subproblems of smaller size
- Step 2 Solve each subproblem recursively by the same algorithm. Use boundary conditions when the size of the subproblem is "minimum" to initialize the recursion
- Step 3 Combine the solutions of the subproblems to obtain the solution of the original problem



(a)

$N = LM$



(b)

15

Row-Wise Data Mapping

Row-wise $n = Ml + m$ $N = LM$

l \ m	0	1	2	…	$M-1$
0	$x(0)$	$x(1)$	$x(2)$	…	$x(M-1)$
1	$x(M)$	$x(M+1)$	$x(M+2)$	…	$x(2M-1)$
2	$x(2M)$	$x(2M+1)$	$x(2M+2)$	…	$x(3M-1)$
	⋮	⋮	⋮	⋯	⋮
$L-1$	$x((L-1)M)$	$x((L-1)M+1)$	$x((L-1)M+2)$	…	$x(LM-1)$

16

Column-Wise Data Mapping

Column-wise $n = l + mL$ $N = LM$

l \ m	0	1	2	…	$M-1$
0	$x(0)$	$x(L)$	$x(2L)$	…	$x((M-1)L)$
1	$x(1)$	$x(L+1)$	$x(2L+1)$	⋯	$x((M-1)L+1)$
2	$x(2)$	$x(L+2)$	$x(2L+2)$	…	$x((M-1)L+2)$
	⋮	⋮	⋮	⋯	⋮
$L-1$	$x(L-1)$	$x(2L-1)$	$x(3L-1)$	⋯	$x(LM-1)$

17

Divide-and-Conquer DFT Decomposition

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad N^2 \text{ OPs}$$

$N = LM$

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

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

$$W_N^{Nmp} = 1$$

$$W_N^{Mpl} = W_{N/M}^{pl} = W_L^{pl}$$

$$W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}$$

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

$$F(l,q) = \sum_{m=0}^{M-1} x(l,m) W_M^{mq} \qquad LM^2 \text{ OPs}$$

$$G(l,q) = W_N^{lq} F(l,q) \qquad LM \text{ OPs}$$

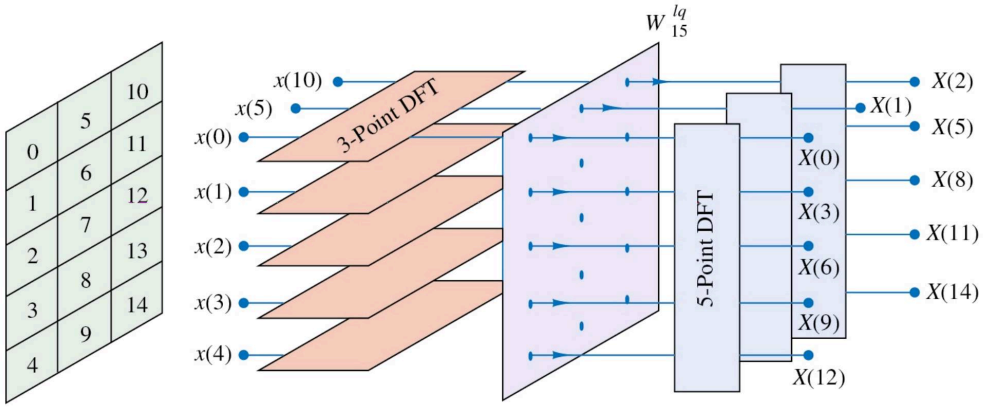
$$X(p,q) = \sum_{l=0}^{L-1} G(l,q) W_L^{lp} \qquad ML^2 \text{ OPs}$$

$N(M+L)$ OPs

OP = One Complex Multiplication and Addition

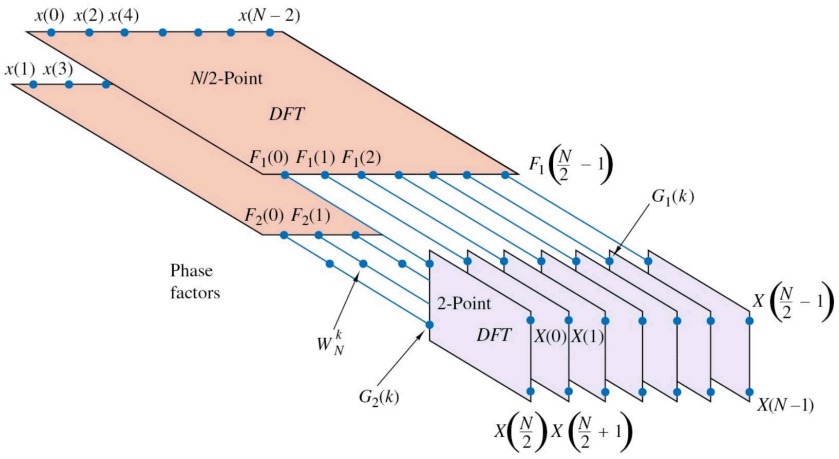
18

Example of 3×5 Decomposition



19

Radix-2 FFT Algorithms



20

Decimation in Time (DIT) Radix-2 FFT

$$X(k) = \underbrace{\sum_{n=0}^{N-1} x(n)W_N^{kn}}_{DFT_N} = \sum_{n=\text{even}} x(n)W_N^{kn} + \sum_{n=\text{odd}}^{N-1} x(n)W_N^{kn}, \qquad k = 0,1,\dots,N-1$$

$$n = 2m \qquad n = 2m+1 \qquad m = 0,1,\dots,\frac{N}{2}-1$$

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

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

$$X(k) = \underbrace{\sum_{m=0}^{(N/2)-1} a(m)W_{N/2}^{km}}_{DFT_{N/2}} + W_N^k \underbrace{\sum_{m=0}^{(N/2)-1} b(m)W_{N/2}^{km}}_{DFT_{N/2}}$$

$$W_N^{k+\frac{N}{2}} = e^{j\pi}W_N^k = -W_N^k$$

$$X(k) = A(k) + W_N^k B(k), \qquad k = 0,1,\dots\frac{N}{2}-1$$

$$X\left(k+\frac{N}{2}\right) = A(k) + W_N^{k+\frac{N}{2}}B(k) = A(k) - W_N^k B(k), \qquad 0,1,\dots,\frac{N}{2}-1$$

21

Computational Complexity

DFT Computation

$$a(n) = x(2n) \xrightarrow[N/2]{\text{DFT}} A(k) = \sum_{n=0}^{(N/2)-1} a(n) W_{N/2}^{kn}$$

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

$$b(n) = x(2n+1) \xrightarrow[N/2]{\text{DFT}} B(k) = \sum_{n=0}^{(N/2)-1} b(n) W_{N/2}^{kn}$$

DFT Merging

$$X(k) = A(k) + W_N^k B(k)$$

$$X\left(k+\frac{N}{2}\right) = A(k) - W_N^k B(k)$$

Holds for every even N!

Merging

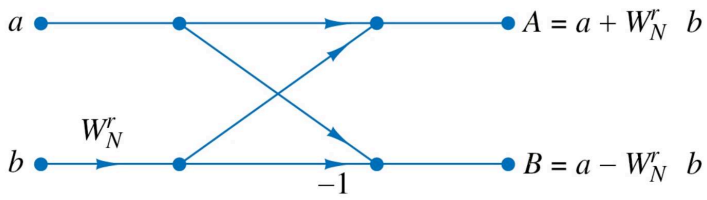
Complexity

$$2\left(\frac{N}{2}\right)^2 + \frac{N}{2} \approx \frac{N^2}{2} \Rightarrow$$

50% reduction

22

Decimation-In-Time Butterfly



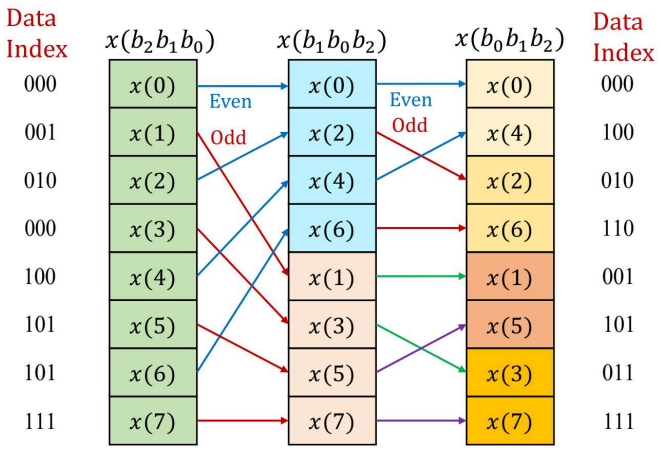
Basic computation flow graph

$$X(k) = A(k) + W_N^k B(k)$$

$$X\left(k+\frac{N}{2}\right) = A(k) - W_N^k B(k)$$

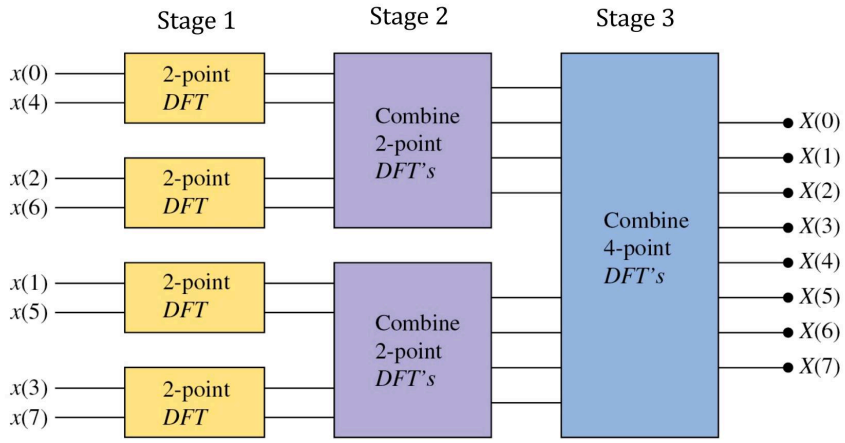
23

Data Shuffling and Bit-Reversal



24

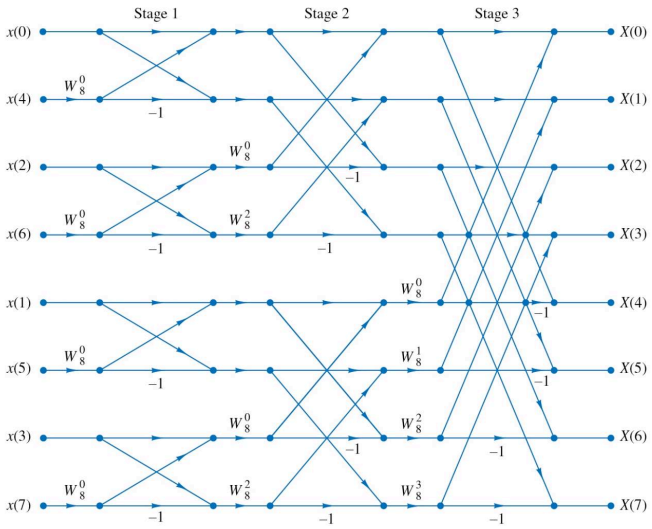
Illustration of DFT Merging



$N/2$ OPs to merge at each stage $\times \log_2 N$ stages$= \frac{N}{2}\log_2 N$

25

Decimation-In-Time Radix-2 FFT Flowgraph



26

Examples of Computational Complexity

Number of Points, N	Complex Multiplications in Direct Computation, N^2	Complex Multiplications in FFT Algorithm, $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,536	1,024	64.0
512	262,144	2,304	113.8
1,024	1,048,576	5,120	204.8

27

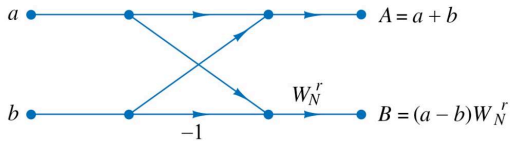
Decimation-in-Frequency Radix-2 Algorithms

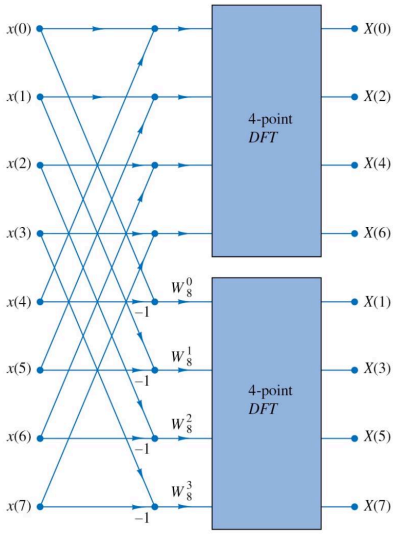
$$X(2\ell) = \sum_{n=0}^{N-1} x(n) W_N^{(2\ell)n} = \sum_{n=0}^{N-1} x(n) W_{N/2}^{\ell n}$$

$$X(2\ell) = \sum_{n=0}^{N/2-1} x(n) W_{N/2}^{(2\ell)n} + \sum_{n=0}^{N/2-1} x\left(n + \frac{N}{2}\right) W_{N/2}^{\ell(n+N/2)}$$

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

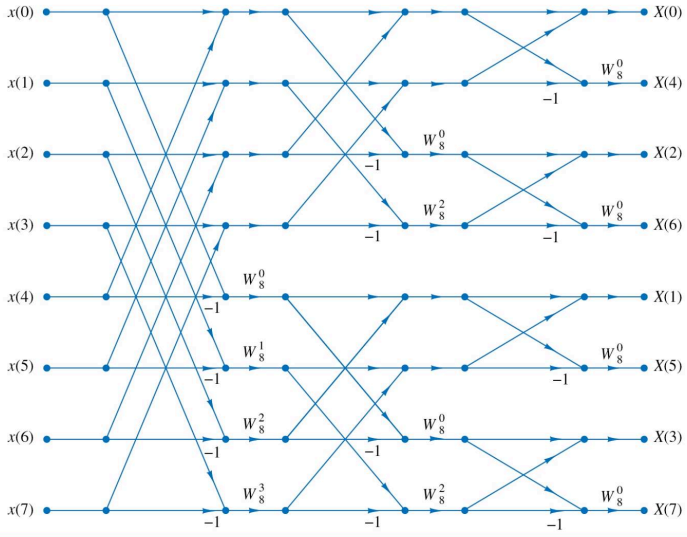
$$X(2\ell + 1) = \sum_{n=0}^{N/2-1} W_N^n [x(n) - x(n + N/2)] W_{N/2}^{\ell n}$$





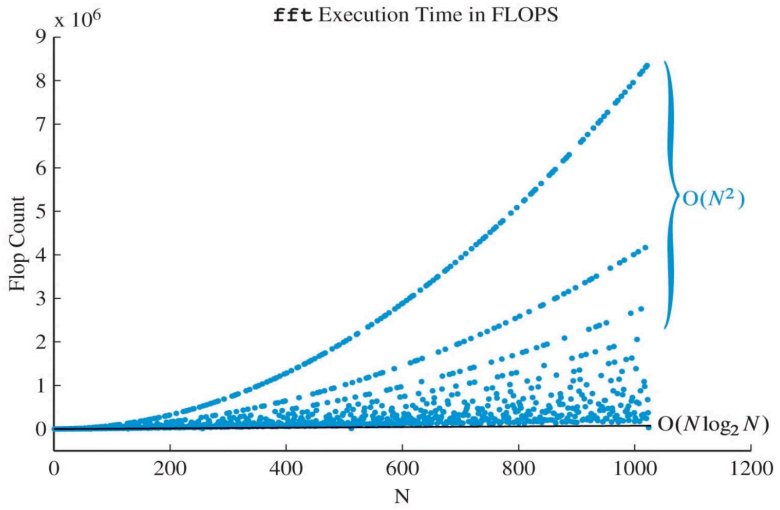
28

Example of DIF Radix-2 Structure



29

FFT Performance Advantage



30

Some Practical Considerations

- Changing the order of the nodes in an FFT flowgraph does not change the resulting DFT coefficients
- If the input and output nodes of each computation butterfly are horizontally adjacent, the computation can be done "in-place" using only one array of N complex memory locations
- The bit-reversed shuffling of the input data is necessary for (or facilitates) in-place computation
- Rearranging the order of the nodes leads to algorithms with
  - Input in normal order and output in bit-reversed order
  - Both input and output in normal order
  - Same geometry for each stage
- Twiddle factors are obtained from look-up table or recursive computation using digital oscillators
- Other FFT algorithms include: Radix 4, Radix 8, Split-Radix, Mixed Radix
- Indexing operations include: Input sorting into bit-reversed order, output sorting into natural order, and accessing the required twiddle factors

31

MATLAB's Native Functions

- The DFT and IDFT are efficiently computed in MATLAB using functions **`fft`** and **`ifft`**, respectively.
- The **`X = fft(x)`** function computes the **`length(x)`** DFT of vector **`x`** in **`X`**. Similarly, **`x = ifft(x)`** computes the **`length(X)`** IDFT of vector **`X`** in **`x`**.
- To compute the DFT of a specific length N, the **`X = fft(x,N)`** invocation is used in which the vector **`x`** is padded with zeros to length **`N`** if it is larger than the length of **`x`**, otherwise **`x`** is truncated to the first N samples.
- The **`x = ifft(X,N)`** is also used in a similar manner.
- MATLAB also provides two additional functions, **`fftshift`** and **`ifftshift`**, which are useful in plotting **`fft`**/**`ifft`** results.
- The **`fftshift`** function shifts the zero-frequency (or DC) component to the center of the spectrum while the **`ifftshift`** function undoes the results of **`fftshift`**.

Summary

- Direct computation of N-point DFT requires $O(N^2)$ operations
- Fast Fourier Transform (FFT) algorithms reduce the computational complexity from $O(N^2)$ to $O(N \log_2 N)$ operations
- The decimation-in-time radix-2 FFT algorithm, which requires $N = 2^v$, is widely used because it is easy to derive, simple to program, and extremely fast (Advice to FFT users: use radix-2 FFT with zero-padding!)
- For many years the time for the computation of FFT algorithms was dominated by multiplications and additions. The performance of FFTs on current computers depends upon the structure of the algorithm, the compiler, and the architecture of the machine
- For applications which do not require all N-DFT coefficients, we can use algorithms based on linear filtering operations (these include Goertzel's algorithm and the chirp transform algorithm) or sparce FFT algorithms
- MATLAB uses "The Fastest Fourier Transform in the West" (**`www.fftw.org`**)

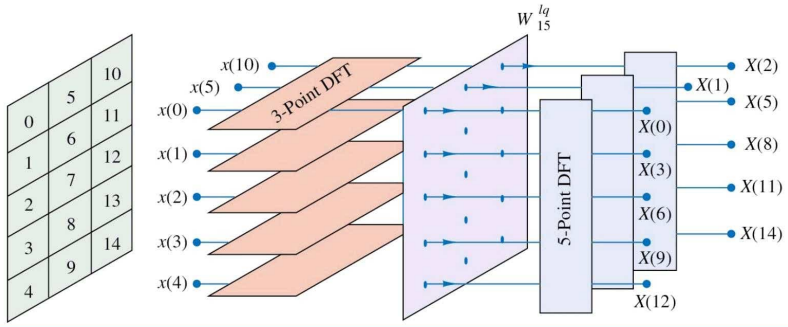
Reading Material and Homework Assignment

- Reading material: Section 8.1 (except 8.1.4, 8.1.5)

- Homework:
  - Suppose you have a function `X = fft(x)` that computes the DFT. Can you use this function to compute the inverse DFT?
  - Study "**`Technical_Note_on_Numerical_Computation_of_CTFT.pdf`**" document on how to use fft functions to numerically compute CTFT.
  - Study "**`Use_of_DFT_and_FFT_in_DTFT.pdf`**" document.
  - Study "**`Technical_Note_on_16-point_Radix-4_FFT.pdf`**" document.

34

Computation of DFT: Fast Fourier Transform Algorithms



35