Fast Fourier Transform

- Direct computation of DFT
- FFT algorithms using divide and conquer
  - Basic idea is to separate the whole DFT into smaller pieces to avoid repetitions, and smaller pieces require much less computation
- FFT algorithms using linear filtering
  - Goertzel algorithm
- Error analysis
- Applications

Direct Computation of DFT

- We have

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

- For each value of $X(k)$ we need N complex multiplications, $N - 1$ additions
- Total N^2 multiplications ($4N^2$ real multiplications), and $N^2 - N$ additions
- And calculation of exponentials. Independent of data, can be pre-calculated.

Why We Can Do It Fast?

- Such a direct computation would be valid for any values of W_N
- In DFT W_N is a nice function with certain properties
  - Symmetry: $W_N^{k+N/2} = -W_N^k$
  - Periodicity: $W_N^{k+N} = W_N^k$
- These allow for fast computation algorithms

Divide and Conquer

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{LM} & \frac{2}{LM} & \cdots & \frac{N-1}{LM} \\ 0 & \frac{2}{LM} & \frac{4}{LM} & \cdots & \frac{2N-2}{LM} \\ & & \cdots & & \\ 0 & \frac{L}{LM} = \frac{1}{M} & \frac{2L}{LM} = \frac{2}{M} & \cdots & \frac{LN-L}{LM} \\ 0 & \frac{L+1}{LM} = \frac{1}{M} + \frac{1}{LM} & \frac{2L+2}{LM} = \frac{2}{M} + \frac{2}{LM} & \cdots & \frac{(L+1)(N-1)}{LM} = \frac{N-1}{M} + \frac{N-1}{LM} \\ & & \cdots & & \end{array} \times \begin{bmatrix} x[0] \\ x[1] \\ . \\ . \\ x[N-1] \end{bmatrix}$$

Divide and Conquer (Cont.)

- Let us regroup the signal and its DFT so that the repetitions from the previous slide can be exploited
- Regroup $x[n]$ as $x[l,m]$ and $X[k]$ as $X[p,q]$
- We now have

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

- Using $W_N^{Nmp} = 1$, $W_N^{mqL} = W_M^{mq}$, $W_N^{Mpl} = W_L^{pl}$

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

Divide and Conquer: Cost

- Total cost: $N(M + L + 1)$ complex multiplications, $N(M + L - 2)$ complex additions instead of N^2 complex multiplications and $N^2 - N$ complex additions
- Example $N = 10000$, $M = 100$, $L = 100$. Direct computation: 10^8 multiplications, divide and conquer: $198.10^4 \rightarrow$ approximately 50 times savings
- Even further simplifications possible when N can be divided into more number of products of prime numbers

Radix-2 FFT Algorithm

- Special case of divide and conquer where $N = 2^v$
- Our DFT is

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

- But we have $W_N^2 = W_{N/2}$,

$$\begin{aligned} X[k] &= \sum_{m=0}^{(N/2)-1} f_1[m] W_{N/2}^{km} + W_N^k \sum_{m-0}^{(N/2)-1} f_2[m] W_{N/2}^{km} \\ &= F_1[k] + W_N^k F_2[k] \end{aligned}$$

Radix-2 FFT Algorithm (Cont.)

- Utilize the periodicity $F_1[k]$ and $F_2[k]$ with $W_N^{k+N/2} = -W_N k$:

$$X[k] = F_1[k] + W_N^k F_2[k] \qquad k = 0, 1, \ldots, N/2 - 1$$

$$X[k + N/2] = F_1[k] - W_N^k F_2[k] \qquad k = 0, 1, \ldots, N/2 - 1$$

- Computation cost: $2(N/2)^2 + N/2 = N^2/2 + N/2$ multiplications, about half reduction in multiplication number

- We can even further divide each of the DFT's by two since $N = 2^v$ resulting in $(N/2)\log_2(N)$ multiplications in total

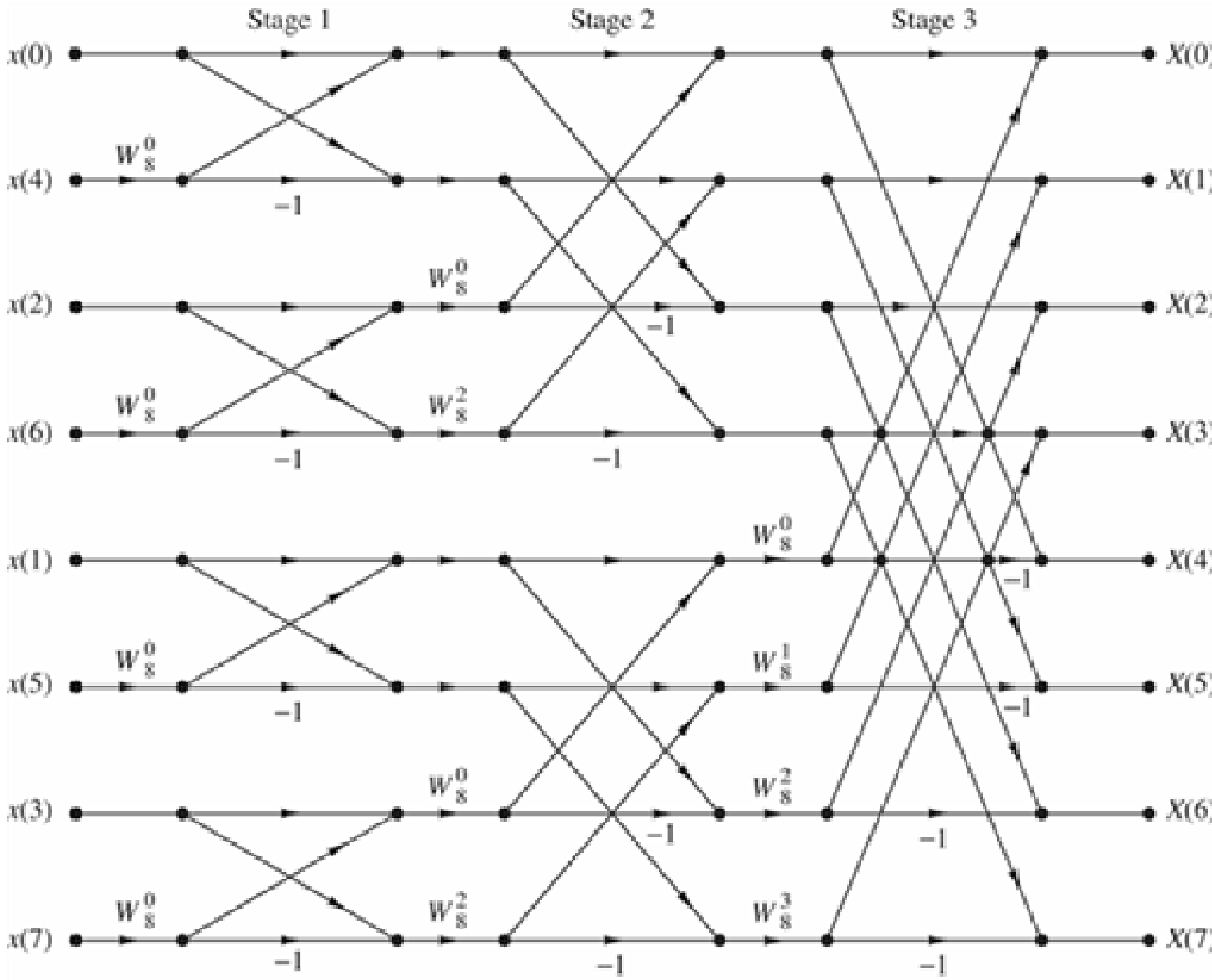

Figure 8.1.6 Eight-point decimation-in-time FFT algorithm.

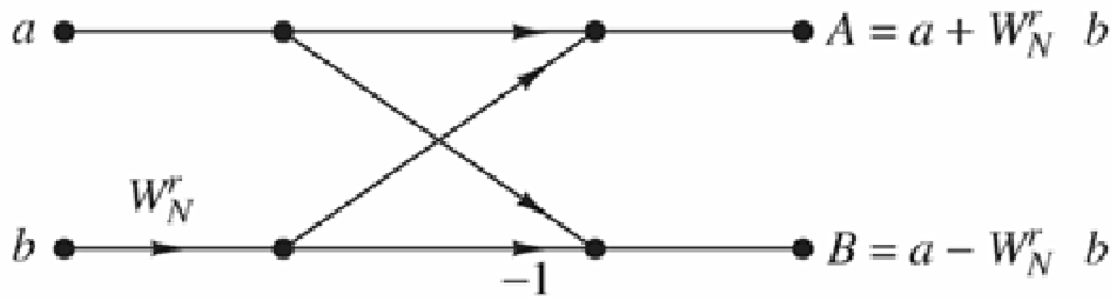

Figure 8.1.7 Basic butterfly computation in the decimation-in-time FFT algorithm.

FFT using linear filtering approaches

- DFT can be seen as a filtering operation with filter having the impulse response W_N^{kn}
- FFT is more efficient when the number of DFT points is large
- Linear filtering methods are more efficient when the number of DFT points is small
- Goertzel Algorithm

Goertzel Algorithm

- Let us modify original DFT by multiplying it with $W_N^{-kN} = 1$:

$$X[k] = y_k[N] = W_N^{-kN} \sum_{m=0}^{N-1} x[m] W_N^{km} = \sum_{m=0}^{N-1} x[m] W_N^{-k(n-m)}$$

- This is a convolution sum which can be computed using a recursive relation:

$$y_k[n] = W_N^{-k} y_k[n-1] + x[n]$$

- We need N multiplications to reach $y_k[N]$
- Assume we need only one value of the DFT then N complex multiplications is sufficient
- More efficient when number of points needed is less than $log_2(N)$

Error Analysis

- We can use only finite number of bits when calculating the DFT
- This causes round-off or quantization errors
- Assuming that we use b bits, errors can be in the range $[-0.5^{(b+1)}, 0.5^{(b+1)}]$
- Let us assume that the error is uniformly distributed: $\sigma_e^2 = \frac{0.5^{2b}}{12}$
- Remember we have $4N^2$ real multiplications
- Assuming uncorrelated errors we have the total variance

$$\sigma_2 = 4N\sigma_e^2 = \frac{N}{3}0.5^{2b}$$

- More bits smaller error of course..
- Another error source is scaling to prevent overflowing

Error Analysis (Cont.)

- In DFT we have seen that error variance depends on the number of multiplications
- So: does FFT (with smaller number of multiplications) result in smaller error
- Is this heaven: simpler calculation AND less error?
- No!, the error is the same as DFT
- Expected since mathematically FFT and DFT are identical
- What happens is that the errors in multiplications are no longer independent..

Applications

- Of course any place where DFT is used
- Linear filtering: convolution
- Correlation: time reverse one sequence and calculate the convolution