

并行计算

2022年春

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.1.1 离散傅里叶变换(DFT)

11.1.2 DFT的顺序代码

11.1.3 串行FFT递归算法

11.1.4 串行FFT非递归算法

11.2 并行FFT算法

离散傅里叶变换(DFT)

- 定义

给定向量 $A=(a_0, a_1, \dots, a_{n-1})^T$, DFT 将 A 变换为 $B=(b_0, b_1, \dots, b_{n-1})^T$

即
$$b_j = \sum_{k=0}^{n-1} a_k \omega^{kj} \quad 0 \leq j \leq n-1$$

这里 $\omega = e^{2\pi i/n}$ 为 n 次单位元根, $i = \sqrt{-1}$; 写成矩阵形式为

$$\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.1.1 离散傅里叶变换(DFT)

11.1.2 DFT的顺序代码

11.1.3 串行FFT递归算法

11.1.4 串行FFT非递归算法

11.2 并行FFT算法

DFT的顺序代码

- 代码1

```
for j=0 to n-1 do
  b[j]=0
  for k=0 to n-1 do
    b[j]=b[j]+ $\omega^{k*j}$ a[k]
  end for
end for
```

注：代码1需要计算 ω^{k*j}

- 代码2

```
w= $\omega^0$ 
for j=0 to n-1 do
  b[j]=0, s= $\omega^0$ 
  for k=0 to n-1 do
    b[j]=b[j]+s*a[k]
    s=s*w
  end for
  w=w* $\omega$ 
end for
```

代码2的复杂度为 $O(n^2)$

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.1.1 离散傅里叶变换(DFT)

11.1.2 DFT的顺序代码

11.1.3 串行FFT递归算法

11.1.4 串行FFT非递归算法

11.2 并行FFT算法

串行FFT递归算法 (1)

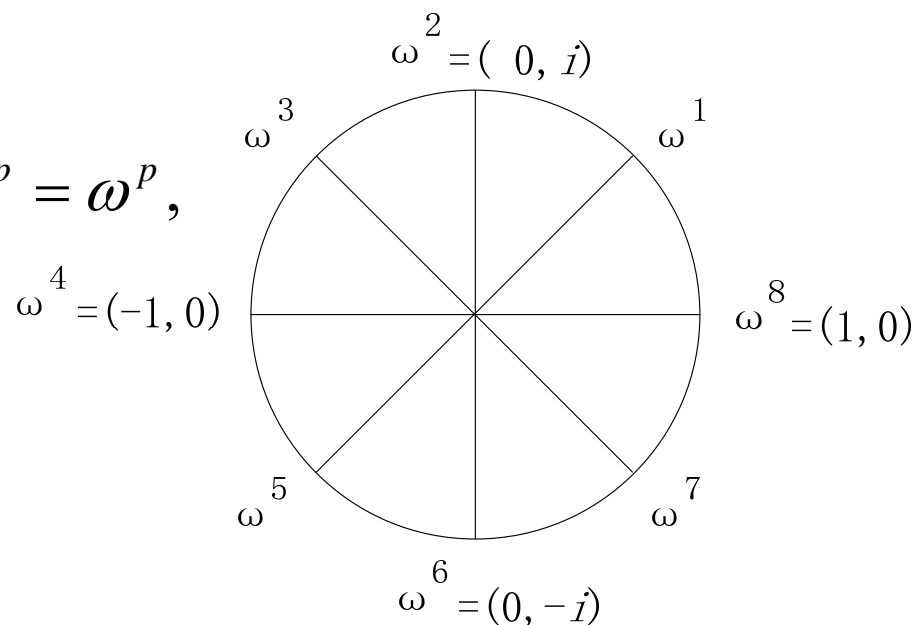
- 蝶式递归计算原理

令 $\tilde{\omega} = e^{2\pi i/(n/2)}$ 为 $n/2$ 次单位元根, 则有 $\tilde{\omega} = \omega^2$.

将 b 向量的偶数项 $(b_0, b_2, \dots, b_{n-2})^T$ 和奇数项 $(b_1, b_3, \dots, b_{n-1})^T$ 分别记为 $(b'_0, b'_1, \dots, b'_{\frac{n}{2}-1})^T$ 和 $(b''_0, b''_1, \dots, b''_{\frac{n}{2}-1})^T$

注意推导中反复使用

$$\omega^n = 1, \omega^{n/2} = -1, \omega^{ln} = 1, \omega^{sn+p} = \omega^p,$$



串行FFT递归算法（2）

$$\begin{aligned}\text{偶数时: } b'_l &= b_{2l} = \sum_{k=0}^{n-1} \omega^{2lk} a_k \\ &= a_0 + \omega^{2l} a_1 + \omega^{4l} a_2 + \cdots + \omega^{2l(\frac{n}{2}-1)} a_{\frac{n}{2}-1} + \\ &\quad a_{\frac{n}{2}} + \omega^{2l} a_{\frac{n}{2}+1} + \omega^{4l} a_{\frac{n}{2}+2} + \cdots + \omega^{2l(\frac{n}{2}-1)} a_{n-1} \\ &= (a_0 + a_{\frac{n}{2}}) + \omega^{2l} (a_1 + a_{\frac{n}{2}+1}) + \omega^{4l} (a_2 + a_{\frac{n}{2}+2}) + \\ &\quad \cdots + \omega^{2l(\frac{n}{2}-1)} (a_{\frac{n}{2}-1} + a_{n-1}) \\ &= (a_0 + a_{\frac{n}{2}}) + \tilde{\omega}^l (a_1 + a_{\frac{n}{2}+1}) + \tilde{\omega}^{2l} (a_2 + a_{\frac{n}{2}+2}) + \\ &\quad \cdots + \tilde{\omega}^{l(\frac{n}{2}-1)} (a_{\frac{n}{2}-1} + a_{n-1}) \\ &= \sum_{k=0}^{\frac{n}{2}-1} \tilde{\omega}^{kl} (a_k + a_{\frac{n}{2}+k}) \quad l = 0, 1, \dots, \frac{n}{2} - 1\end{aligned}$$

因此,向量 $(b_0, b_2, \dots, b_{n-2})^T$ 是 $(a_0 + a_{\frac{n}{2}}, a_1 + a_{\frac{n}{2}+1}, \dots, a_{\frac{n}{2}-1} + a_{n-1})^T$ 的DFT

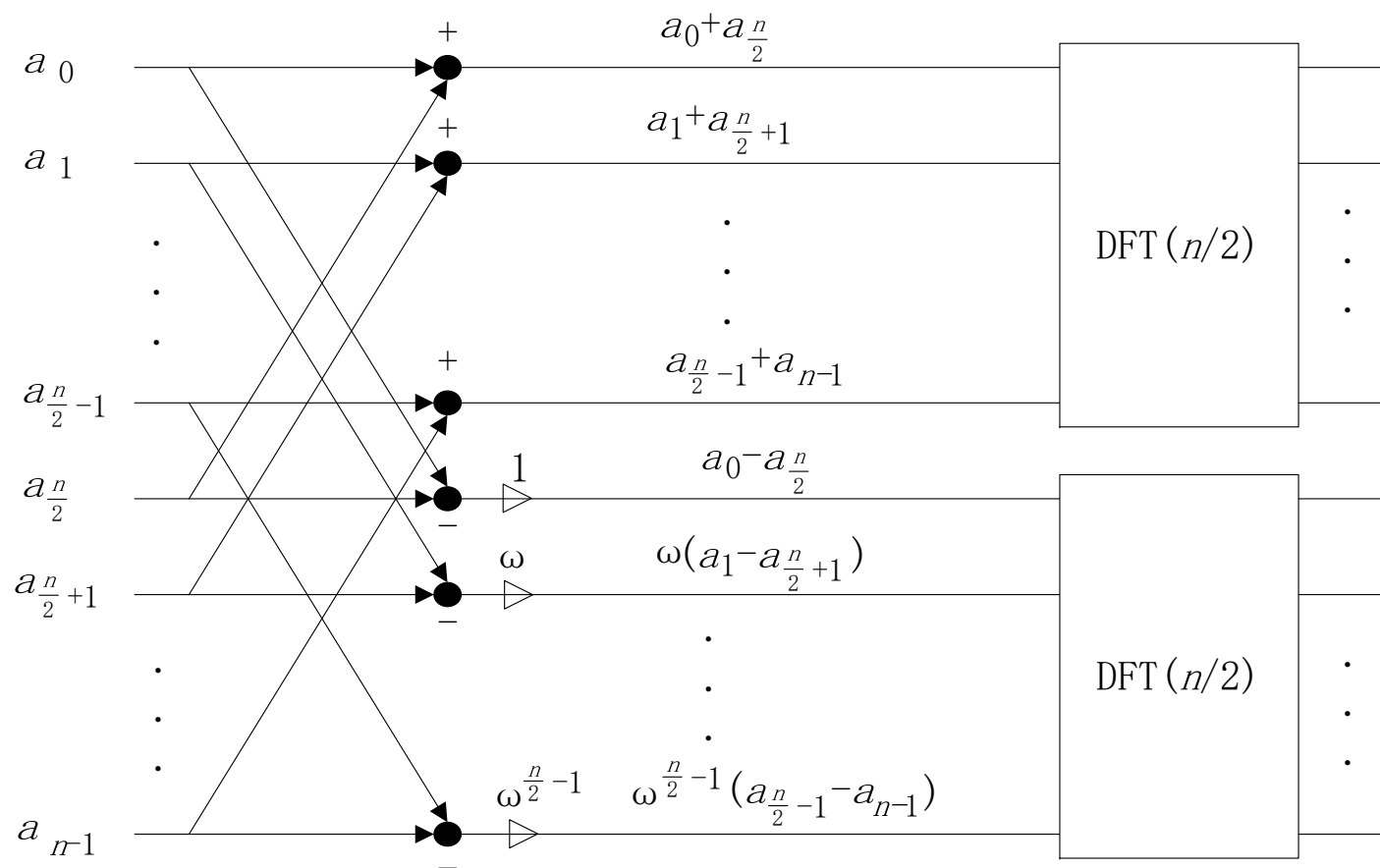
串行FFT递归算法 (3)

$$\begin{aligned}
 \text{奇数时: } b_l'' = b_{2l+1} &= \sum_{k=0}^{n-1} \omega^{(2l+1)k} a_k \\
 &= a_0 + \omega^{2l+1} a_1 + \omega^{2(2l+1)} a_2 + \cdots + \omega^{\left(\frac{n}{2}-1\right)(2l+1)} a_{\frac{n}{2}-1} + \\
 &\quad \omega^{\frac{n}{2}(2l+1)} a_{\frac{n}{2}} + \omega^{\left(\frac{n}{2}+1\right)(2l+1)} a_{\frac{n}{2}+1} + \cdots + \omega^{(n-1)(2l+1)} a_{n-1} \\
 &= a_0 + \omega^{2l} \omega a_1 + \omega^{4l} \omega^2 a_2 + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} a_{\frac{n}{2}-1} - \\
 &\quad a_{\frac{n}{2}} - \omega^{2l} \omega a_{\frac{n}{2}+1} - \cdots - \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} a_{n-1} \\
 &= (a_0 - a_{\frac{n}{2}}) + \omega^{2l} \omega (a_1 - a_{\frac{n}{2}+1}) + \omega^{4l} \omega^2 (a_2 - a_{\frac{n}{2}+2}) + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} (a_{\frac{n}{2}-1} - a_{n-1}) \\
 &= (a_0 - a_{\frac{n}{2}}) + \tilde{\omega}^l \omega (a_1 - a_{\frac{n}{2}+1}) + \tilde{\omega}^{2l} \omega^2 (a_2 - a_{\frac{n}{2}+2}) + \cdots + \tilde{\omega}^{l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} (a_{\frac{n}{2}-1} - a_{n-1}) \\
 &= \sum_{k=0}^{\frac{n}{2}-1} \tilde{\omega}^{kl} \omega^k (a_k - a_{\frac{n}{2}+k}) \quad l = 0, 1, \dots, \frac{n}{2}-1
 \end{aligned}$$

因此, 向量 $(b_1, b_3, \dots, b_{n-1})^T$ 是 $((a_0 - a_{\frac{n}{2}}), \omega(a_1 - a_{\frac{n}{2}+1}), \dots, \omega^{\frac{n}{2}-1}(a_{\frac{n}{2}-1} - a_{n-1}))^T$ 的DFT

串行FFT递归算法（4）

- FFT的蝶式递归计算图(由计算原理推出)



串行FFT递归算法 (5)

特别地, $n=8$ 的FFT蝶式计算图(展开的)

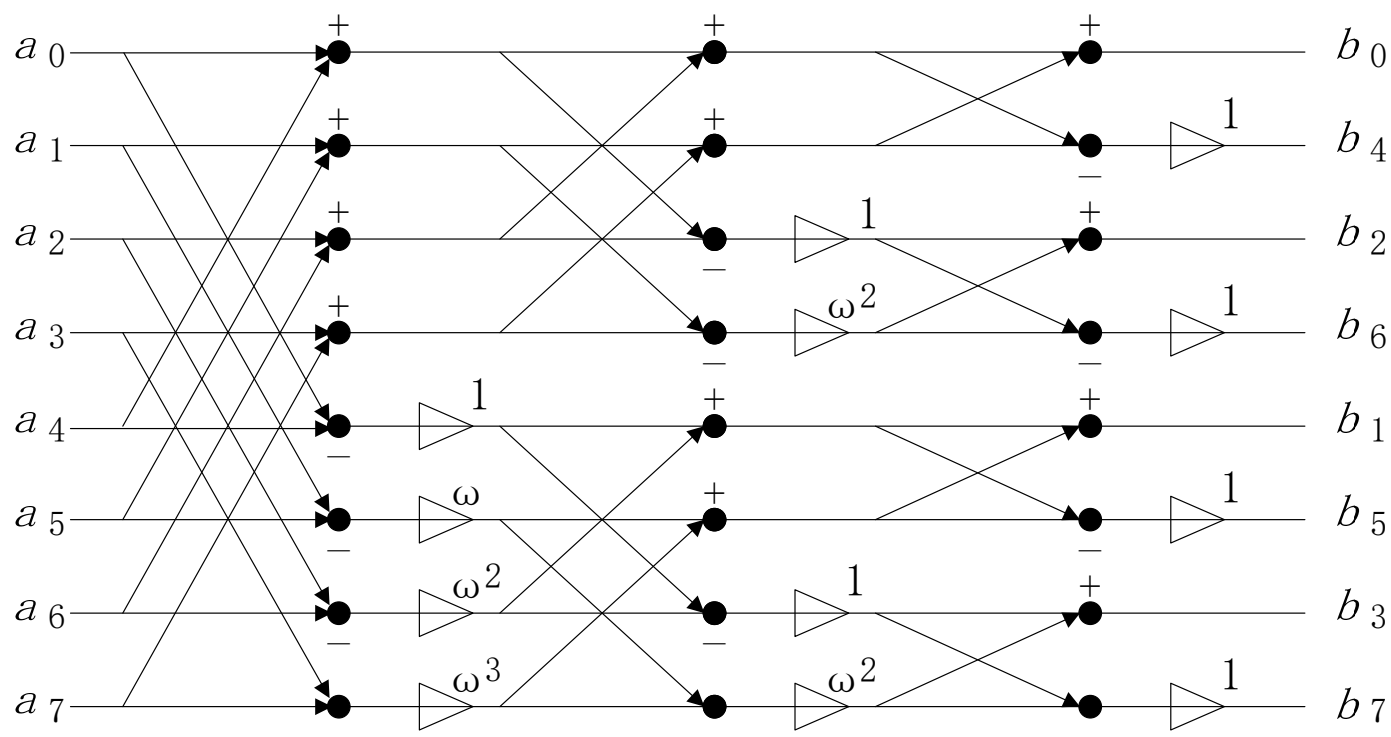


图11.4

串行FFT递归算法 (6)

- SISD上的FFT分治递归算法

输入: $A = (a_0, a_1, \dots, a_{n-1})$; 输出: $B = (b_0, b_1, \dots, b_{n-1})$

Procedure RFFT(A, B)

begin

if $n=1$ then $b_0 = a_0$ else

(1) RFFT($a_0, a_2, \dots, a_{n-2}, u_0, u_1, \dots, u_{n/2-1}$)

(2) RFFT($a_1, a_3, \dots, a_{n-1}, v_0, v_1, \dots, v_{n/2-1}$)

(3) $z = 1$

(4) for $j=0$ to $n-1$ do

(4.1) $b_j = u_{j \bmod n/2} + z v_{j \bmod n/2}$

(4.2) $z = z \omega$

endfor

endif

end

注: (1) 算法时间复杂度 $t(n) = 2t(n/2) + O(n)$

解得 $t(n) = O(n \log n)$

(2) 算法原理?

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.1.1 离散傅里叶变换(DFT)

11.1.2 DFT的顺序代码

11.1.3 串行FFT递归算法

11.1.4 串行FFT非递归算法

11.2 并行FFT算法

串行FFT非递归算法（1）

- 蝶式计算示例

$$\text{当 } n=2 \text{ 时, } \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & \omega \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \Rightarrow \begin{cases} b_0 = a_0 + a_1 \\ b_1 = a_0 - a_1 \end{cases}$$

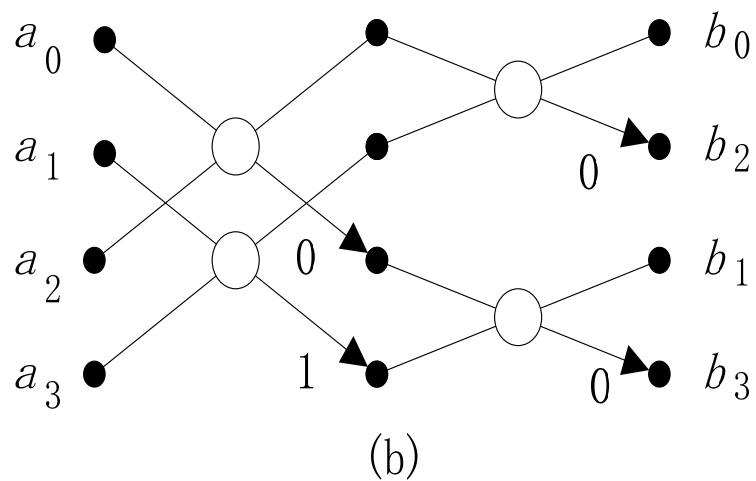
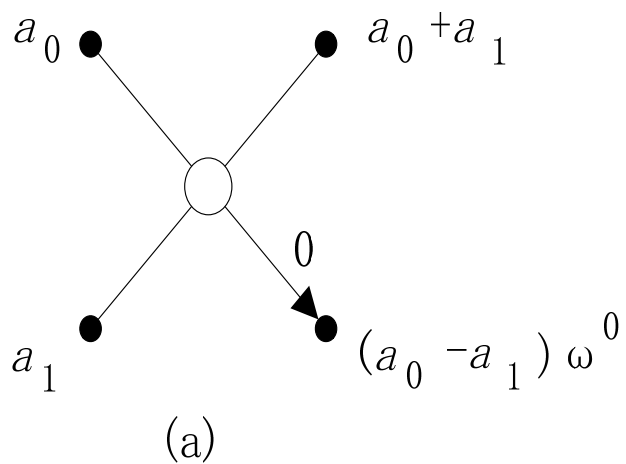
$$\text{当 } n=4 \text{ 时, } \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & -1 & -\omega \\ 1 & -1 & 1 & -1 \\ 1 & -\omega & -1 & \omega \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

对调 b_1 和 b_2 , \Rightarrow

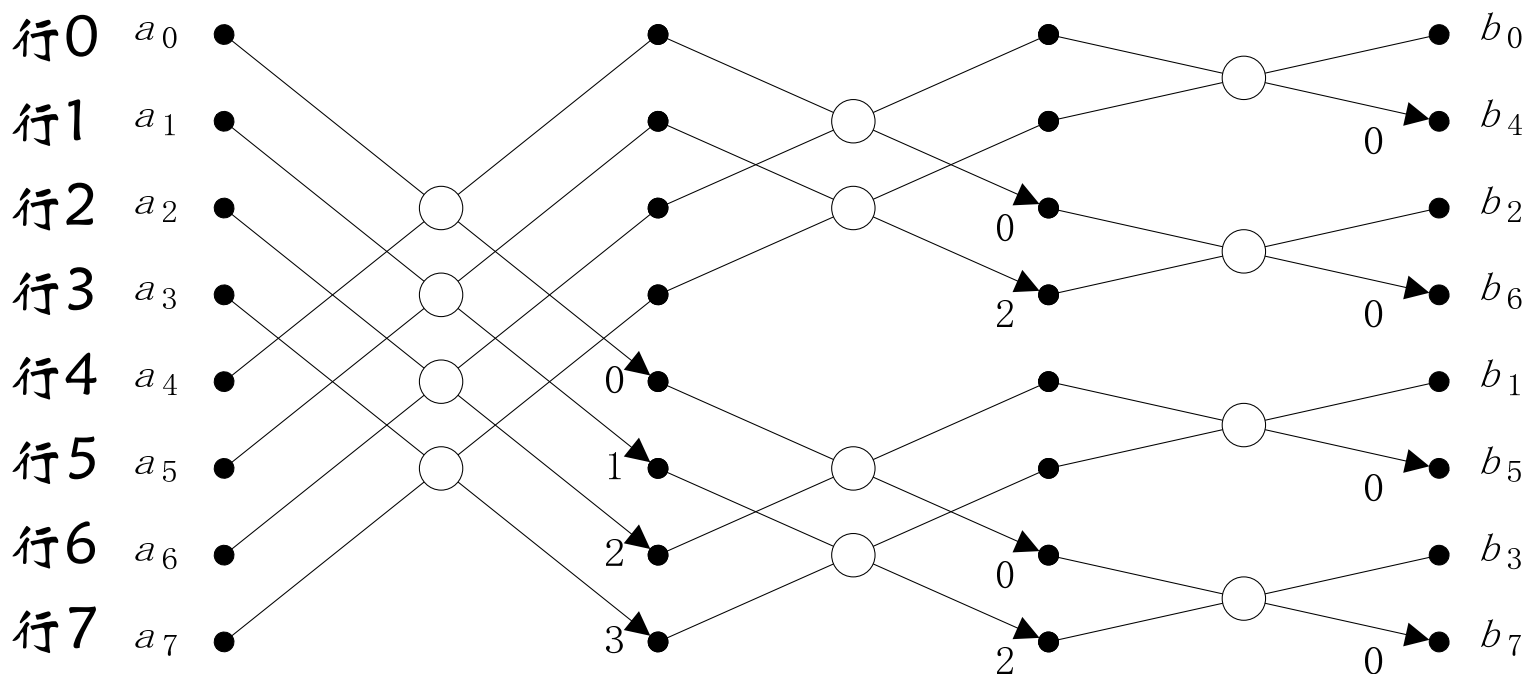
$$\begin{bmatrix} b_0 \\ b_2 \\ b_1 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & \omega & 0 & -\omega \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \Rightarrow \begin{cases} b_0 = (a_0 + a_2) + (a_1 + a_3) \\ b_2 = (a_0 + a_2) - (a_1 + a_3) \\ b_1 = (a_0 - a_2) + (a_1 - a_3)\omega \\ b_3 = (a_0 - a_2) - (a_1 - a_3)\omega \end{cases}$$

串行FFT非递归算法 (2)

- 蝶式计算流图



串行FFT非递归算法 (3)



如: $b_6 = [(a_0 + a_4) - (a_2 + a_6)] - [(a_1 + a_5) - (a_3 + a_7)] \omega^2$

注: ①下行线结点处的权因子的确定问题;

② b_i 的下标确定:取行号的位序反。如, 行3: $3 = (011)_2 \Rightarrow (110)_2 = 6$, \Rightarrow 行3的输出为 b_6

串行FFT非递归算法（4）

- 算法11.1(P266): SISD上FFT迭代算法

输入: $A=(a_0, a_1, \dots, a_{n-1})$; 输出: $B=(b_0, b_1, \dots, b_{n-1})$

如: $n=8$

Begin

(1)for $k=0$ to $n-1$ do $c_k=a_k$ endfor //初始化

$h=2, p=4, q=2, z=\omega^1$

(2)for $h=\log n-1$ to 0 do

$h=1, p=2, q=4, z=\omega^2$

for $k=0$ to $n-1$ do

$h=0, p=1, q=8, z=\omega^4$

(2.1) $p=2^h$ (2.2) $q=n/p$ (2.3) $z=\omega^{q/2}$ //先算出 $z=\omega^1$,以后每次 $z=z \times z$

(2.4)if ($k \bmod p = k \bmod 2p$) then // (i)和(ii)配对执行

(i) $c_k = c_k + c_{k+p}z$

(ii) $c_{k+p} = (c_k - c_{k+p})z^{k \bmod p}$

endif

endfor

endfor

(3)for $k=0$ to $n-1$ do $b_k=c_{r(k)}$ endfor

// $r(k)$ 为 k 的位序反

End

算法时间复杂度: $T(n)=O(n \log n)$

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.2 并行FFT算法

11.2.1 SIMD-MC²上的FFT算法

11.2.2 SIMD-BF上的FFT算法

SIMD-MC²上的FFT算法 (1)

- 算法描述

n 个处理器组成 $n^{1/2} \times n^{1/2}$ 的方阵, 处理器以行主序编号

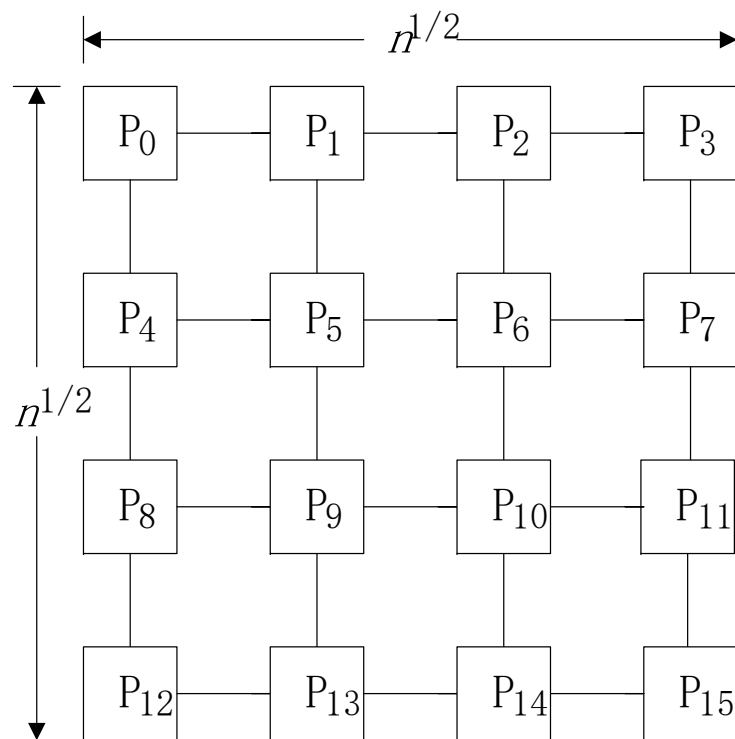


图11.5

SIMD-MC²上的FFT算法 (2)

- 算法11.3(P270): 输入: a_k 处于 P_k 中; 输出 b_k 处于 P_k 中

如: $n=16$

Begin

(1)for $k=0$ to $n-1$ par-do $c_k=a_k$ endfor

$h=3, p=8, q=2, z=\omega^8$

(2)for $h=\log n-1$ to 0 do

$h=2, p=4, q=4, z=\omega^4$

for $k=0$ to $n-1$ par-do

$h=1, p=2, q=8, z=\omega^2$

(2.1) $p=2^h$ (2.2) $q=n/p$ (2.3) $z=\omega^p$ //先算出 $\omega^{n/2}$,以后每次 $z=z^{1/2}$

$h=0, p=1, q=16, z=\omega^1$

(2.4)if ($k \bmod p = k \bmod 2p$) then par-do //满足条件的处理器同时做

(i) $c_k=c_k+c_{k+p}z^{r(k) \bmod q}$

//(i)和(ii)同时执行

(ii) $c_{k+p}=c_k-c_{k+p}z^{r(k) \bmod q}$

endif

endfor

endfor

(3)for $k=0$ to $n-1$ par-do $b_k=c_{r(k)}$ endfor

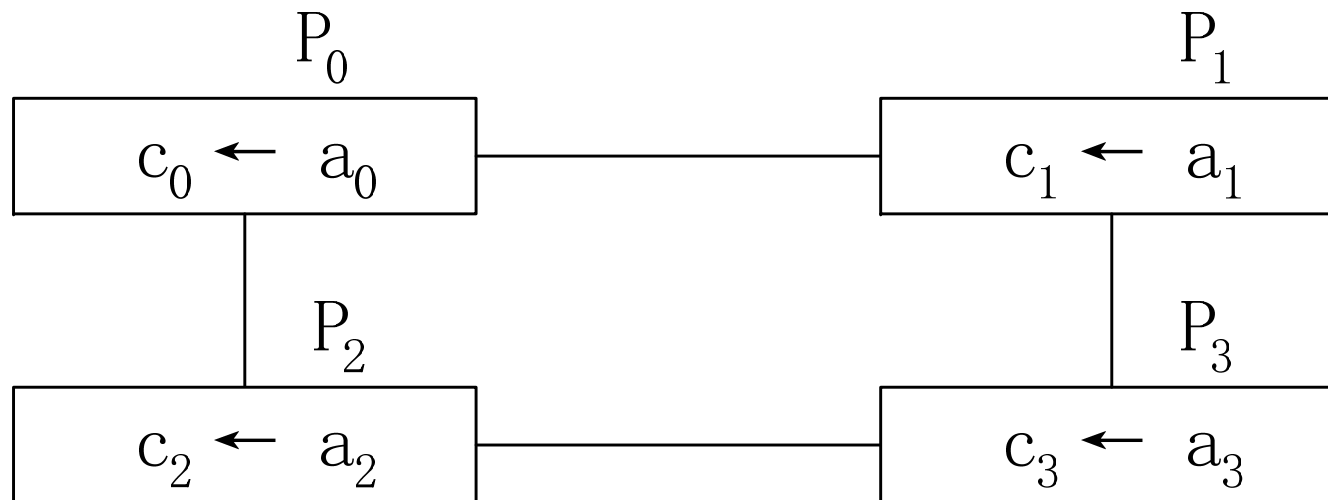
// $r(k)$ 为 k 的位序反

End

SIMD-MC²上的FFT算法 (3)

- 示例: P272例11.5, $n=4$

第(1)步:



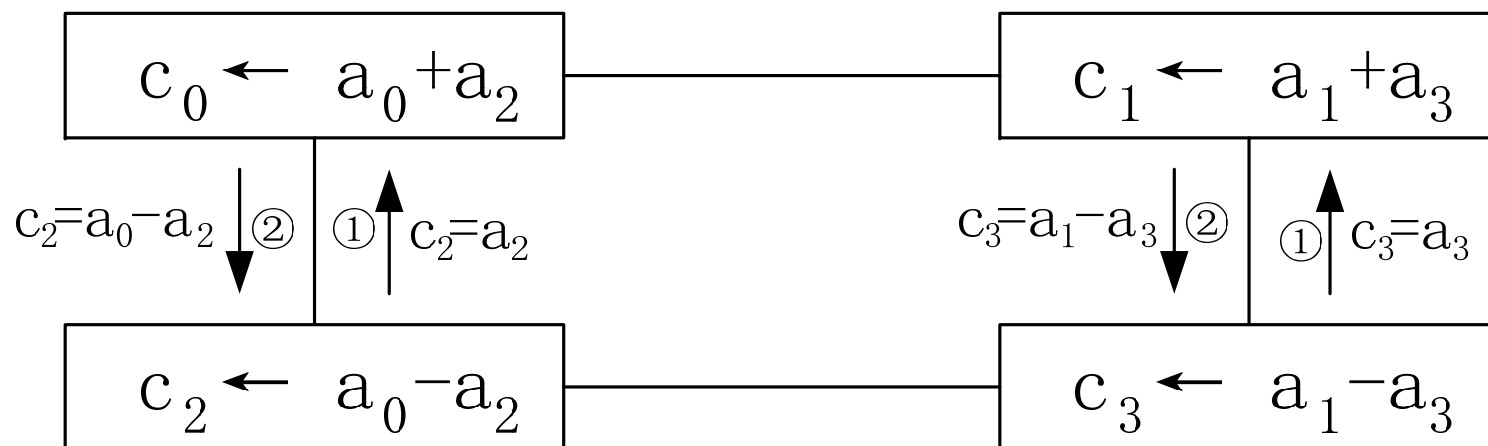
SIMD-MC²上的FFT算法 (4)

第(2)步:

第1次迭代(h=1): $p=2, q=2, z=\omega^2$

满足 $k \bmod 2 = k \bmod 4$ 的处理器为 P_0 和 P_1 , 同时计算

$$\begin{aligned} P_0: c_0 &= c_0 + (\omega^2)^0 c_2 = a_0 + a_2 & P_1: c_1 &= c_1 + (\omega^2)^0 c_3 = a_1 + a_3 \\ c_2 &= c_0 - (\omega^2)^0 c_2 = a_0 - a_2 & c_3 &= c_1 - (\omega^2)^0 c_3 = a_1 - a_3 \end{aligned}$$



SIMD-MC²上的FFT算法 (5)

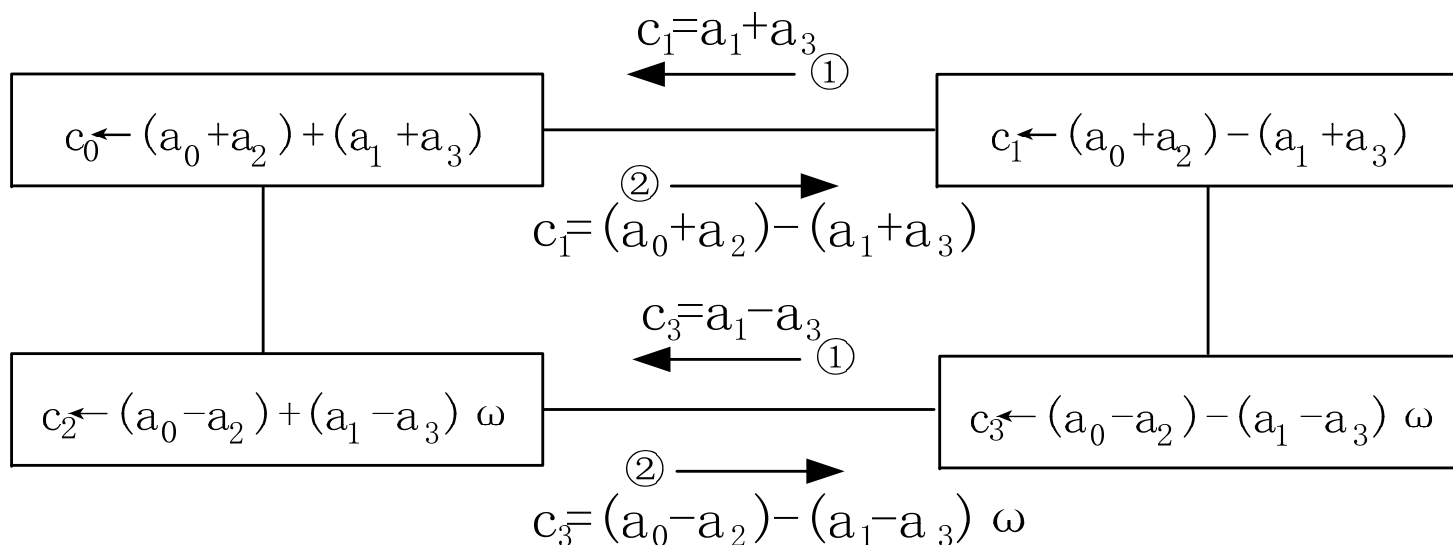
第(2)步:

第2次迭代(h=0): p=1, q=4, z=ω

满足k mod 1 = k mod 2的处理器为P₀和P₂, 同时计算

$$P_0: c_0 = c_0 + \omega^0 c_1 = (a_0 + a_2) + (a_1 + a_3) \quad P_2: c_1 = c_1 + \omega^1 c_3 = (a_0 + a_2) + (a_1 + a_3) \omega$$

$$c_1 = c_0 - \omega^0 c_1 = (a_0 + a_2) - (a_1 + a_3) \quad c_3 = c_1 - \omega^1 c_3 = (a_0 + a_2) + (a_1 + a_3) \omega$$



SIMD-MC²上的FFT算法 (6)

第(3)步: $b_0=c_0, b_1=c_2, b_2=c_1, b_3=c_3$

- 算法分析

- 计算时间: $t_{\text{comp}}=O(\log n)$

- 选路时间: t_{routing} : 只涉及(2.4)和(3)

(2.4): $O(n^{1/2})$

(3): $O(n^{1/2})$

综上, 当 n 较大时 $t(n)=O(n^{1/2})$

第十一章 快速傅里叶变换

11.1 快速傅里叶变换

11.2 并行FFT算法

11.2.1 SIMD-MC²上的FFT算法

11.2.2 SIMD-BF上的FFT算法

SIMD-BF上的FFT算法 (1)

- 蝶形网络

- 处理器布局

有 $k+1$ 层, 每层有 $n=2^k$ 个处理器, 共有 $n(1+\log n)$ 个处理器

第 r 行第 i 列的处理器记为 $P_{r,i}$, $i=(a_1, a_2, \dots, a_k)_2$

- 互连方式

$P_{r,i}$ 与上层 $P_{r-1,i}$, $P_{r-1,j}$ 相连, 这里 i 的第 r 位为0

$P_{r,j}$ 与上层 $P_{r-1,i}$, $P_{r-1,j}$ 相连, 这里 j 与 i 仅在第 r 位不同

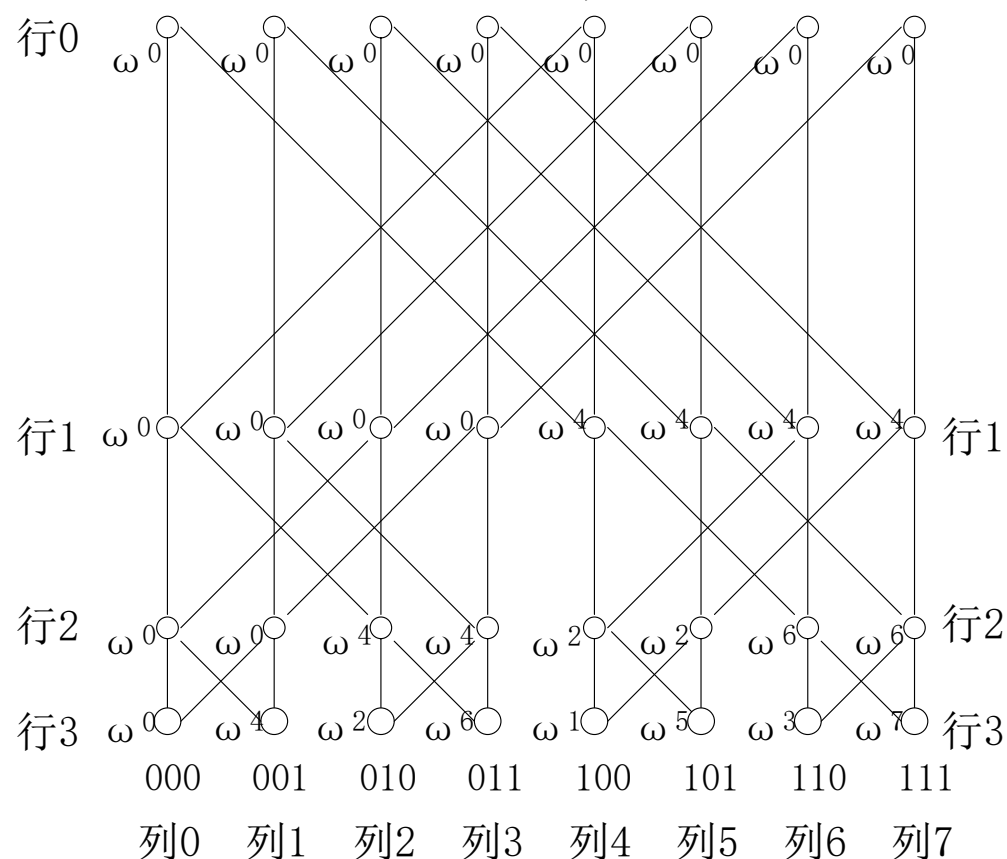
- 权因子 ω 在BF网络中的计算方法

$P_{r,i}$ 中 ω 的指数为 $j=\exp(r,i)$

这里 $\exp(r,i)=(a_r, \dots, a_1, \underbrace{0, \dots, 0}_{k-r})$ //即 i 的前 r 位取位序反, 再后补0

SIMD-BF上的FFT算法 (2)

- 示例: $n=8$ 的BF网络表示



r, i 与上层 $P_{r-1,i}, P_{r-1,j}$ 相连, 这里 i 的第 r 位为0

$P_{r,j}$ 与上层 $P_{r-1,i}, P_{r-1,j}$ 相连, 这里 j 与 i 仅在第 r 位不同

SIMD-BF 上的 FFT 算法 (3)

- 算法描述: P273 算法 11.4

- 算法分析

- 时间分析

第(1)步时间: $O(1)$

(2.1)和(2.2)的计算时间为 $O(1)$, (假定 $\omega^{\exp(r,i)}$ 已计算好)

(2.1)和(2.2)的选路时间为 $O(1)$ \Rightarrow 第(2)步时间: $O(\log n)$

所以 $t(n) = O(\log n)$, $p(n) = n(1 + \log n)$, $c(n) = O(n \log^2 n)$

$S_p(n) = O(n)$, $E_p(n) = O(1/\log n)$ // 本算法的综合指标是较好的

- $\omega^{\exp(r,i)}$ 的计算

初始时: $P_{k,i}$ 读入 $\omega^{\exp(k,i)}$, $k = \log n$

若 $P_{r+1,i}$ 已有 $\omega^{\exp(r+1,i)}$, 则 $P_{r,i}$ 中的 $\omega^{\exp(r,i)} = \omega^{2\exp(r+1,i)}$

所以, 经过 $\log n$ 步就可以计算出每个 $\omega^{\exp(r,i)}$