并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

概要

- 第二篇 并行算法
  - 第九章 稠密矩阵运算
  - 第十章 线性方程组的求解
  - **第十一章 快速傅里叶变换**
  - 第十二章 数值计算的基本支撑技术

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的串行代码
  - 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,⋯,a_{n-1})^T,DFT将A变换为B=(b_0,b_1,⋯,b_{n-1})^T

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

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

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

DFT的串行代码

- 代码1

```
for j=0 to n-1 do
    b[j]=0
    for k=0 to n-1 do
        b[j]=b[j]+ω^(k*j)a[k]
    end for
end for
```

注：代码1需要计算ω^{k*j}

- 代码2

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

复杂度为$O(n^2)$

串行FFT递归算法

- 蝶式递归计算原理（按a展开）

$$
\begin{aligned}
b_j &= \sum_{k=0}^{n-1} \omega^{kj} a_k \\
&= \sum_{k=0}^{n/2-1} \omega^{2kj} a_{2k} + \sum_{k=0}^{n/2-1} \omega^{(2k+1)j} a_{(2k+1)} \\
&= \sum_{k=0}^{n/2-1} \omega^{2kj} a_{2k} + \omega^{j} \sum_{k=0}^{n/2-1} \omega^{2kj} a_{(2k+1)}
\end{aligned}
$$

串行FFT递归算法

- SISD上的FFT分治递归算法

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

```
Procedure RFFT(A, B)
begin
    if n=1 then b_0=a_0 else
        (1)RFFT(a_0,a_2,…,a_{n-2}, u_0,u_1,…,u_{n/2-1})
        (2)RFFT(a_1,a_3,…,a_{n-1}, v_0,v_1,…,v_{n/2-1})
        (3)z=1
        (4)for j=0 to n-1 do
               (4.1)b_j=u_{j mod n/2}+zv_{j mod n/2}
               (4.2)z=zω
          endfor
    endif
end
```

主定理: $T(n)=aT\left(\frac{n}{b}\right)+c(n^d)$

$$\begin{cases} \boxed{T(n)=\Theta(n^d log_2(n)) \quad \text{if } (a=b^d)} \\ T(n)=\Theta(n^d) \quad \text{if } (a<b^d) \\ T(n)=\Theta(n^{log_b(a)}) \quad \text{if } (a>b^d) \end{cases}$$

算法时间复杂度 t(n)=2t(n/2)+O(n)

解得 t(n)=O(nlogn)

串行FFT递归算法

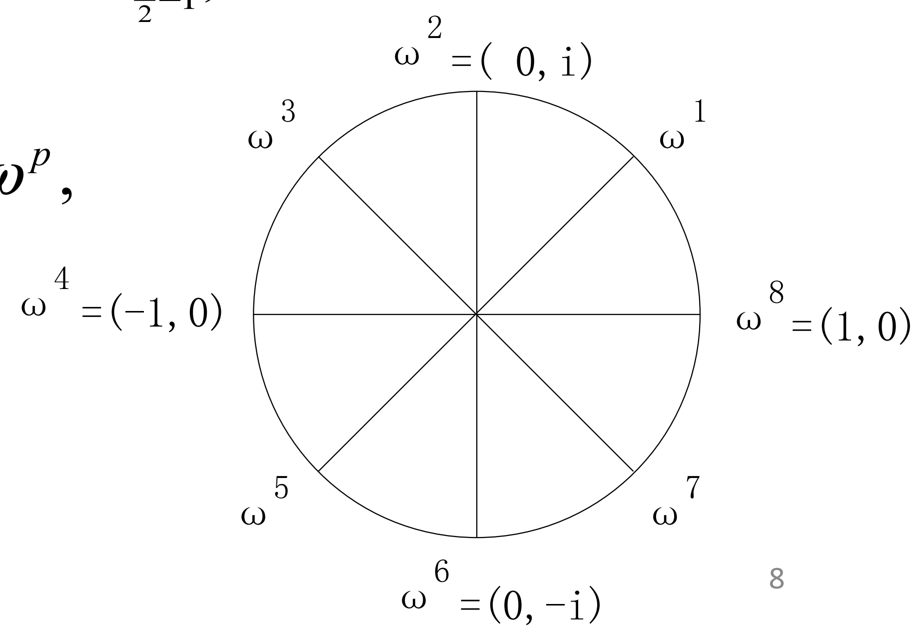
- 蝶式递归计算原理（按b展开）

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 $(b_0, b_2, ..., b_{n-2})^T$ 和奇数项 $(b_1, b_3, ..., b_{n-1})^T$ 分别

记为 $(b'_0, b'_1, ..., b'_{\frac{n}{2}-1})^T$ 和 $(b''_0, b''_1, ..., b''_{\frac{n}{2}-1})^T$

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



串行FFT递归算法

偶数时：$b'_l = b_{2l} = \sum_{k=0}^{n-1} \omega^{2lk} a_k$

$$
\begin{aligned}
&= a_0 + \omega^{2l} a_1 + \omega^{4l} a_2 + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} a_{\frac{n}{2}-1} + \\
&\quad a_{\frac{n}{2}} + \omega^{2l} a_{\frac{n}{2}+1} + \omega^{4l} a_{\frac{n}{2}+2} + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} a_{n-1} \\
&= (a_0 + a_{\frac{n}{2}}) + \omega^{2l}(a_1 + a_{\frac{n}{2}+1}) + \omega^{4l}(a_2 + a_{\frac{n}{2}+2}) + \\
&\quad \cdots + \omega^{2l\left(\frac{n}{2}-1\right)}(a_{\frac{n}{2}-1} + a_{n-1}) \\
&= (a_0 + a_{\frac{n}{2}}) + \tilde{\omega}^{l}(a_1 + a_{\frac{n}{2}+1}) + \tilde{\omega}^{2l}(a_2 + a_{\frac{n}{2}+2}) + \\
&\quad \cdots + \tilde{\omega}^{l\left(\frac{n}{2}-1\right)}(a_{\frac{n}{2}-1} + a_{n-1}) \\
&= \sum_{k=0}^{\frac{n}{2}-1} \tilde{\omega}^{kl}(a_k + a_{\frac{n}{2}+k}) \qquad l = 0,1,\cdots,\tfrac{n}{2}-1
\end{aligned}
$$

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

串行FFT递归算法

奇数时：$b_l'' = b_{2l+1} = \sum_{k=0}^{n-1} \omega^{(2l+1)k} a_k$

$$= a_0 + \omega^{2l+1} a_1 + \omega^{2(2l+1)} a_2 + \cdots + \omega^{\left(\frac{n}{2}-1\right)(2l+1)} a_{\frac{n}{2}-1} +$$

$$\omega^{\frac{n}{2}(2l+1)} a_{\frac{n}{2}} + \omega^{\left(\frac{n}{2}+1\right)(2l+1)} a_{\frac{n}{2}+1} + \cdots + \omega^{(n-1)(2l+1)} a_{n-1}$$

$$= a_0 + \omega^{2l} \omega a_1 + \omega^{4l} \omega^2 a_2 + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} a_{\frac{n}{2}-1} -$$

$$a_{\frac{n}{2}} - \omega^{2l} \omega a_{\frac{n}{2}+1} - \cdots - \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} a_{n-1}$$

$$= (a_0 - a_{\frac{n}{2}}) + \omega^{2l} \omega (a_1 - a_{\frac{n}{2}+1}) + \omega^{4l} \omega^2 (a_2 - a_{\frac{n}{2}+2}) + \cdots + \omega^{2l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} (a_{\frac{n}{2}-1} - a_{n-1})$$

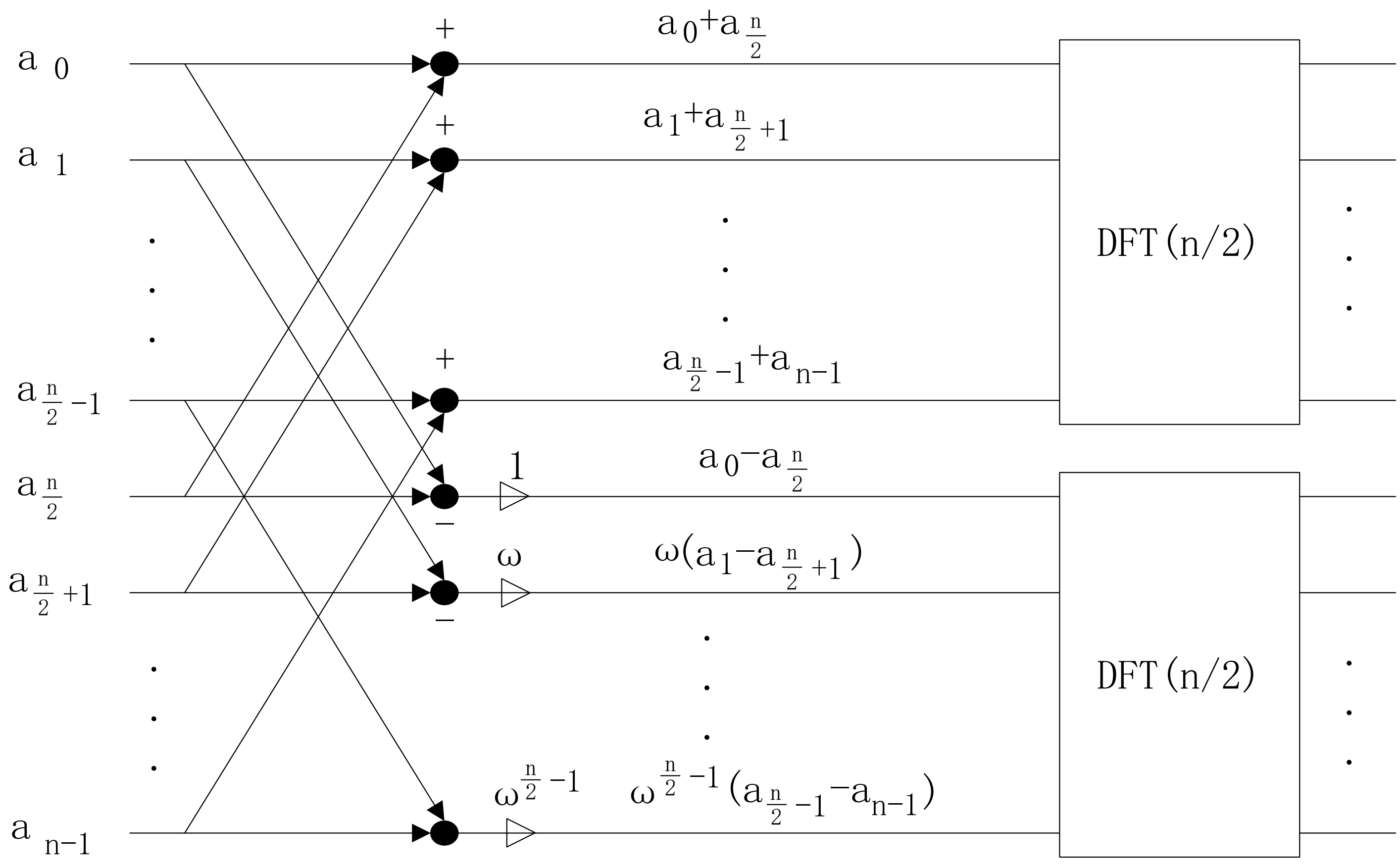
$$= (a_0 - a_{\frac{n}{2}}) + \tilde{\omega}^{l} \omega (a_1 - a_{\frac{n}{2}+1}) + \tilde{\omega}^{2l} \omega^2 (a_2 - a_{\frac{n}{2}+2}) + \cdots + \tilde{\omega}^{l\left(\frac{n}{2}-1\right)} \omega^{\frac{n}{2}-1} (a_{\frac{n}{2}-1} - a_{n-1})$$

$$= \sum_{k=0}^{\frac{n}{2}-1} \tilde{\omega}^{kl} \omega^k (a_k - a_{\frac{n}{2}+k}) \qquad l = 0,1,\cdots,\tfrac{n}{2}-1$$

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

串行FFT递归算法

- FFT的蝶式递归计算图



串行FFT递归算法

- n=8的FFT蝶式计算图

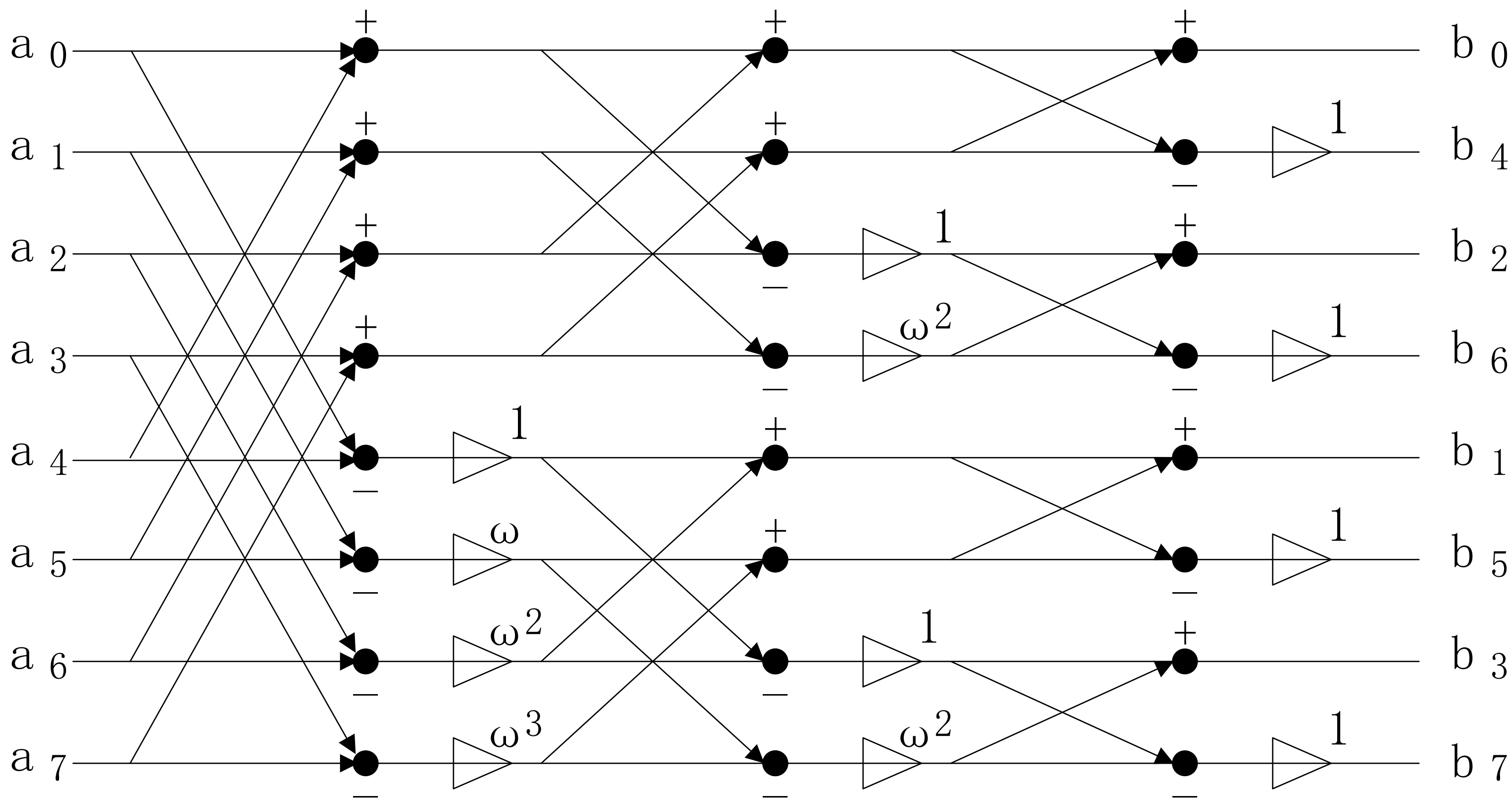


图11.4

串行FFT非递归算法

- 蝶式计算示例

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

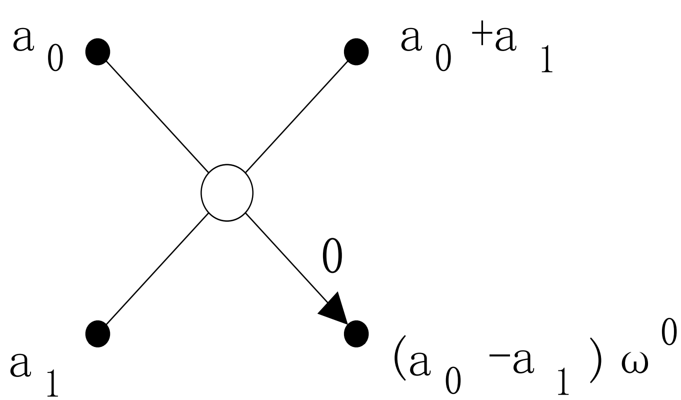
$$\text{当}n=4\text{时},\quad \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix}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

$$\text{对调}b_1\text{和}b_2,\Rightarrow$$

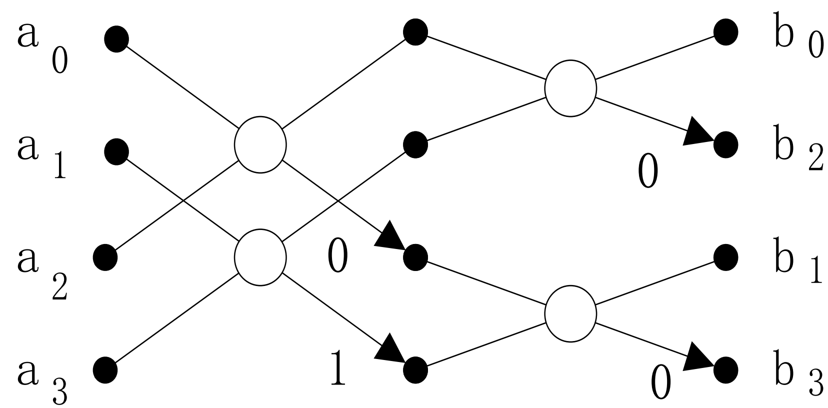
$$\begin{bmatrix} b_0 \\ b_2 \\ b_1 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & \omega & 0 & -\omega \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3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

串行FFT非递归算法

- 蝶式计算流图

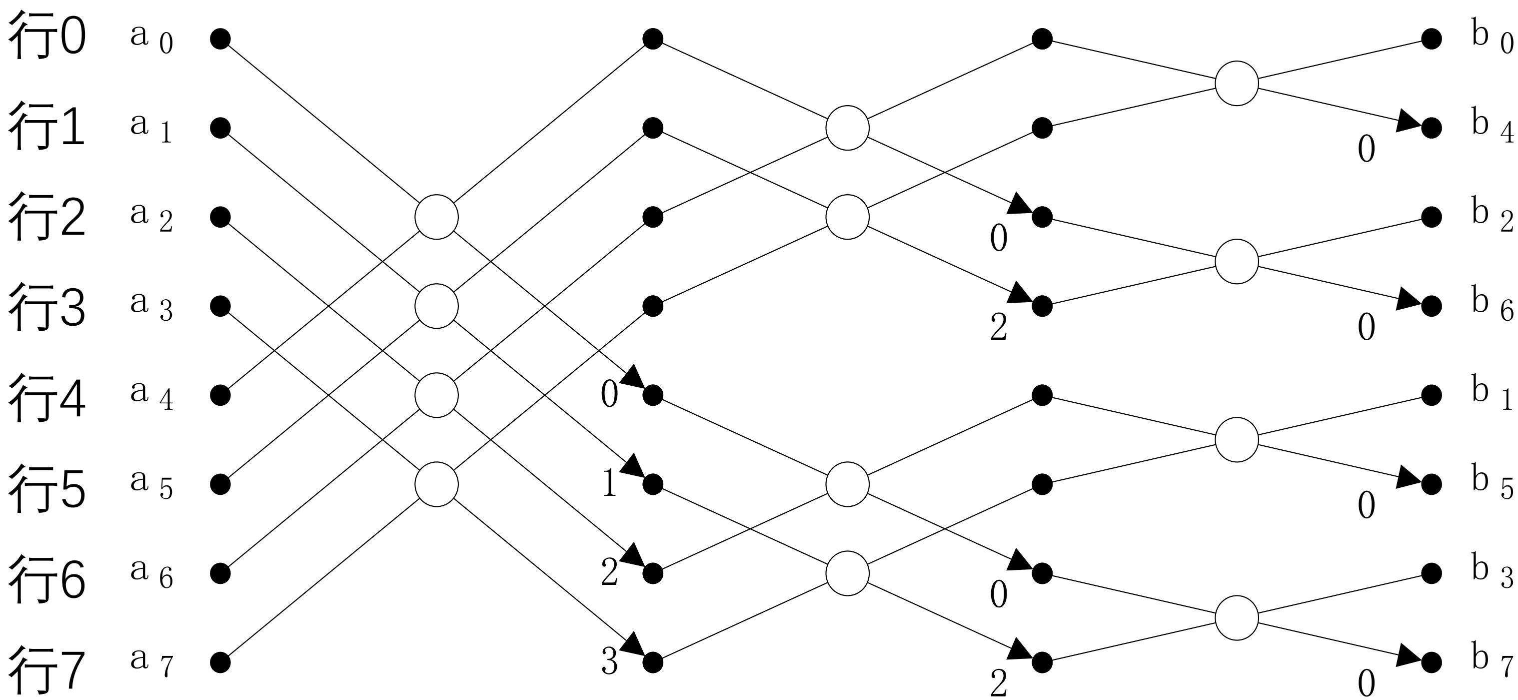


(a)



(b)

串行FFT非递归算法



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ω权因子的确定问题;

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

串行FFT非递归算法

- 算法11.1: SISD上FFT迭代算法

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

Begin

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

(2)for h=logn-1 to 0 do

for k=0 to n-1 do

(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 mod p = k mod 2p) then //注意(i)和(ii)同时执行，读取相同的c_k

(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

例如： n=8

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

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

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

算法时间复杂度：T(n)=O(nlogn)

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-MC^{2}上的FFT算法
    - 11.2.2 SIMD-BF上的FFT算法

SIMD-MC^{2}上的FFT算法

- 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

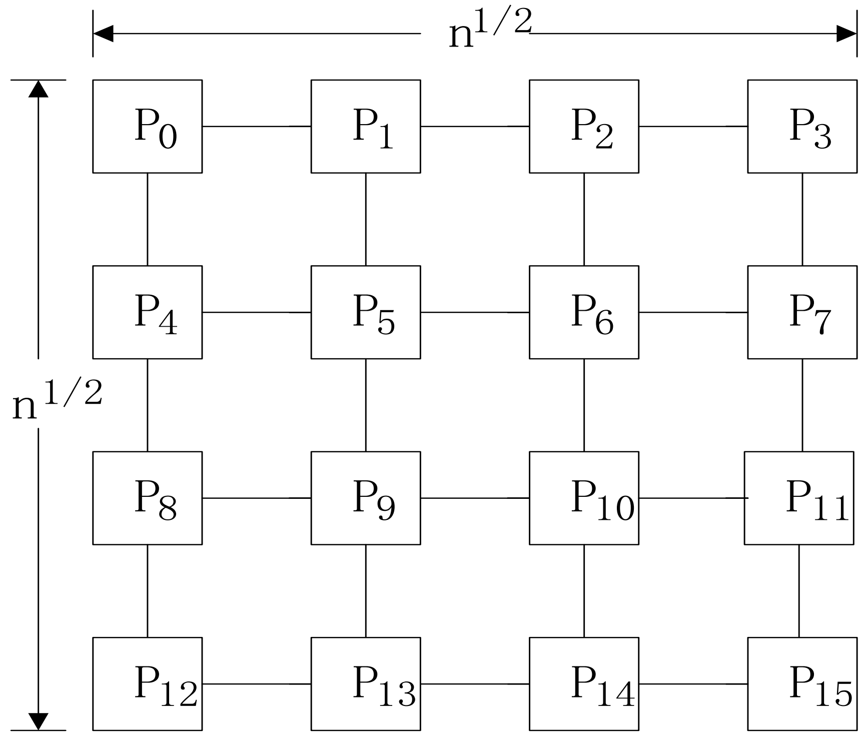


图11. 5

SIMD-MC^{2}上的FFT算法

- 算法11.3 输入: a_k处于P_k中; 输出b_k处于P_k中

例如：n=16

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

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

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

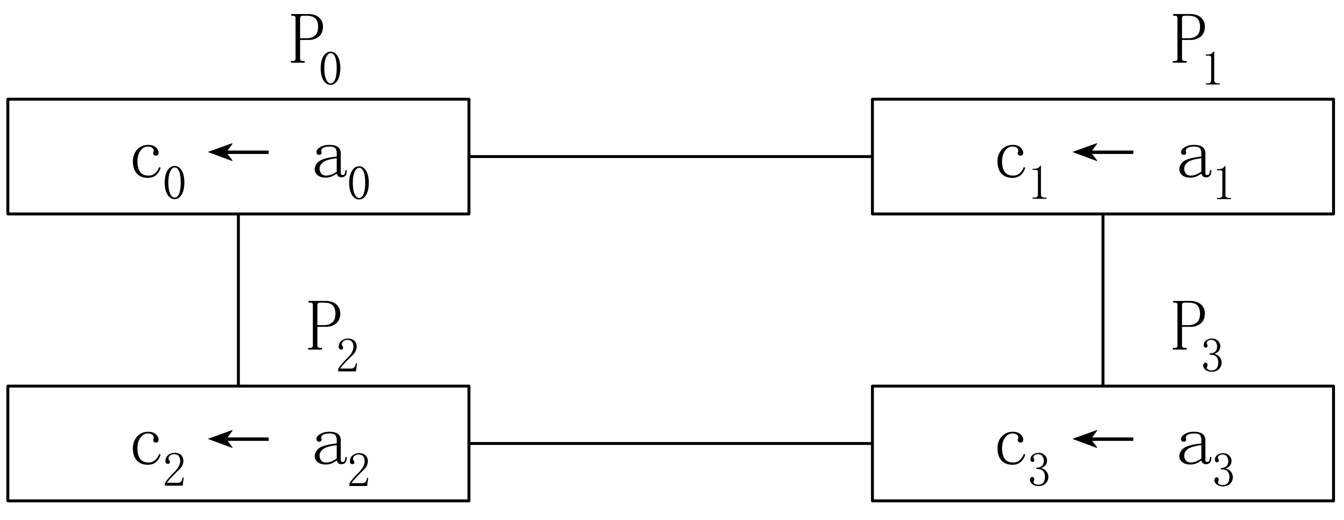
h=0, p=1, q=16, $z=\omega^1$

```
Begin
  (1)for k=0 to n-1 par-do c_k=a_k endfor
  (2)for h=logn-1 to 0 do
        for k=0 to n-1 par-do //并行处理
          (2.1)p=2^h (2.2)q=n/p (2.3)z=ω^p   //先算出ω^{n/2},以后每次z=z^{1/2}
          (2.4)if ( k mod p = k mod 2p) then par-do  //特定处理器执行运算
                 (i) c_k=c_k+c_{k+p}z^{r(k)modq}          确保参与运算的处理器
                 (ii)c_{k+p}=c_k-c_{k+p}z^{r(k)modq}      处于同一行或者同一列
               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-MC^{2}上的FFT算法

- 示例: 例11.5, n=4

第(1)步:



SIMD-MC^{2}上的FFT算法

第(2)步:

第1次迭代(h=1): $p=2,\ q=2,\ z=\omega^2$

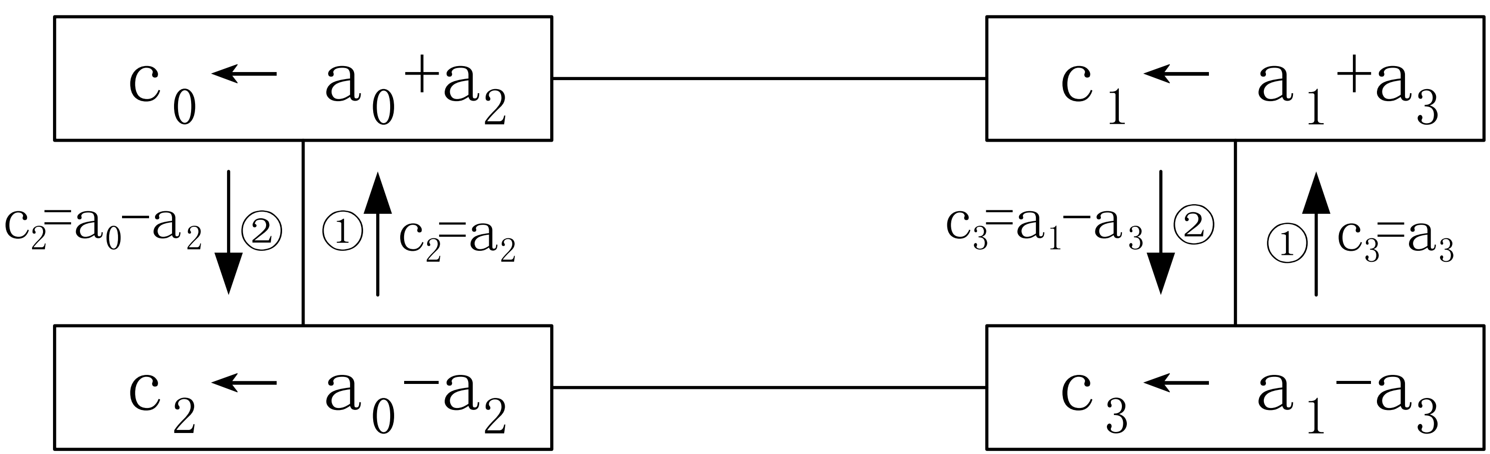
满足k mod 2 = k mod 4的处理器为P_0和P_1, 同时计算

P_0: $c_0=c_0+(\omega^2)^0c_2=a_0+a_2$

$c_2=c_0-(\omega^2)^0c_2=a_0-a_2$

P_1: $c_1=c_1+(\omega^2)^0c_3=a_1+a_3$

$c_3=c_1-(\omega^2)^0c_3=a_1-a_3$



SIMD-MC^{2}上的FFT算法

第(2)步:

第2次迭代(h=0): p=1, q=4, z=ω

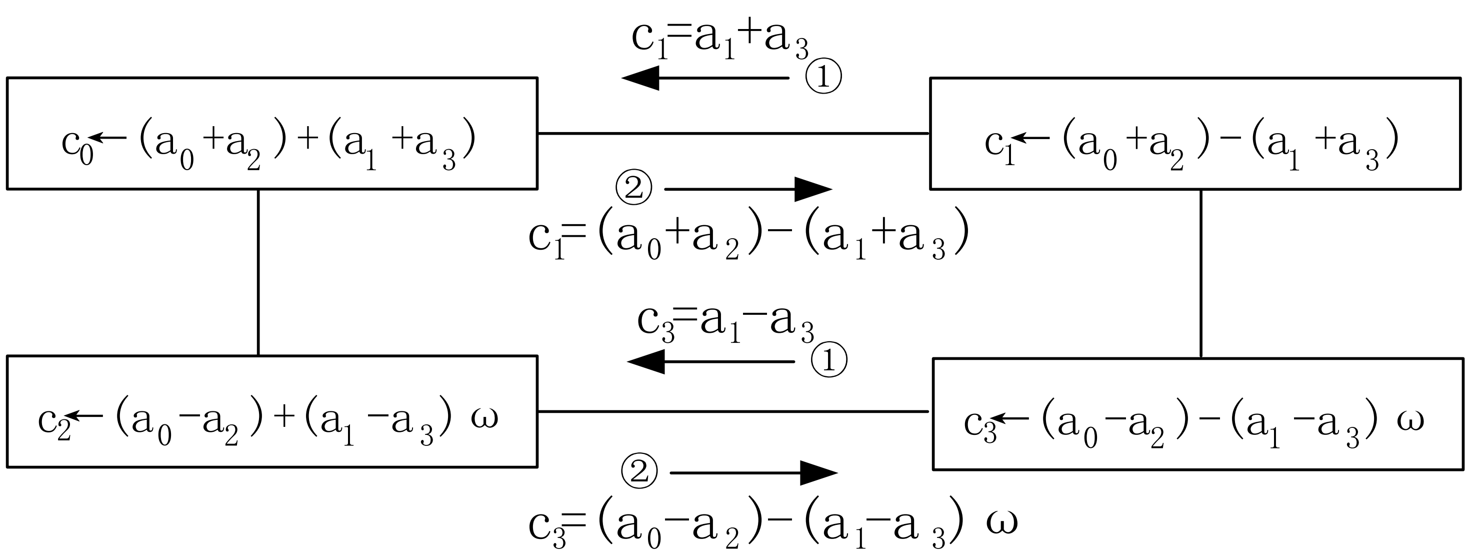
满足k mod 1 = k mod 2的处理器为P_0和P_2, 同时计算

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

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

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

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



SIMD-MC^{2}上的FFT算法

第(3)步: b0=c0, b1=c2, b2=c1, b3=c3

- 算法分析
  - 计算时间: $t_{comp}=O(\log n)$
  - 选路时间: $t_{routing}$: 只涉及(2.4)和(3)

logn次迭代，每次迭代是O(1)计算时间

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

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

综上, 当n较大时$t(n)=O(n^{1/2})$

MC^{2}不太适合FFT

SIMD-BF上的FFT算法

- 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+logn)个处理器

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

  - 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

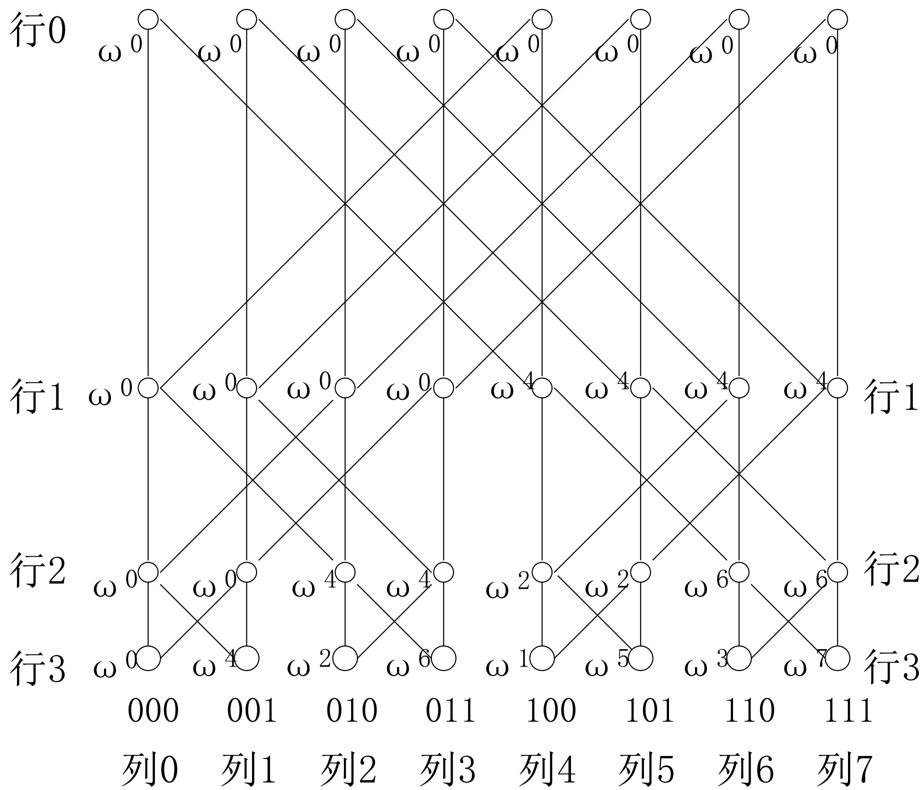
  - 权因子ω在BF网络中的计算方法

$P_{r,i}$中ω的指数为j=exp(r,i)

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

SIMD-BF上的FFT算法

- 示例：n=8的BF网络表示



r,i与上层Pr-1,i, Pr-1,j相连，这里i的第r位为0

Pr,j与上层Pr-1,i, Pr-1,j相连，这里j与i仅在第r位不同

SIMD-BF上的FFT算法

- 算法描述: 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) ==>第(2)步时间: O(logn)

    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$\omega^{exp(r,i)}=\omega^{2exp(r+1,i)}$

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

课后作业4

- 教材P265 分析算法10.6时间复杂度
- 教材P309 11.5

- 提交方式：BB系统
- 截止时间：5月12日23:59