第十一章 快速傅里叶变换

- FFT is one of the most widely used algorithms in science and engineering. It's applications range from signal and data processing and analysis, data compression to the solution of partial differential equations.
- DFT(time series, waveform analysis, linear partial differential equations, convolution, digital signal processing, and image filtering.) $\Theta(n^2)$
- FFT(Cooley and Tukey 1965) $\Theta(n\log n)$

离散傅里叶变换(DFT)

- 定义

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

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

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

ω的乘幂又叫旋转因子

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
```

- 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
```

注：代码1需要计算ω^{k*j}
代码2的复杂度为$O(n^2)$

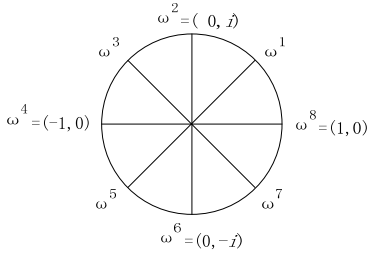
串行FFT递归算法

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

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

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

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

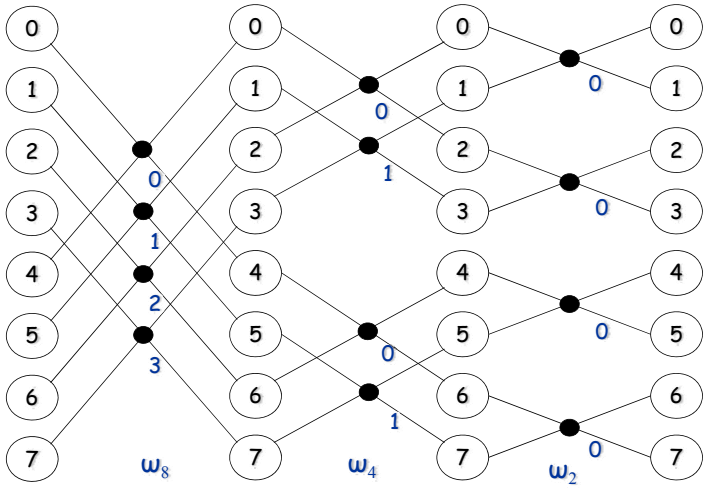
因此，向量$(b_0, b_2, \ldots, b_{n-2})^T$是$(a_0 + a_{\frac{n}{2}}, a_1 + a_{\frac{n}{2}+1}, \ldots, a_{\frac{n}{2}-1} + a_{n-1})^T$的$DFT$

串行FFT递归算法

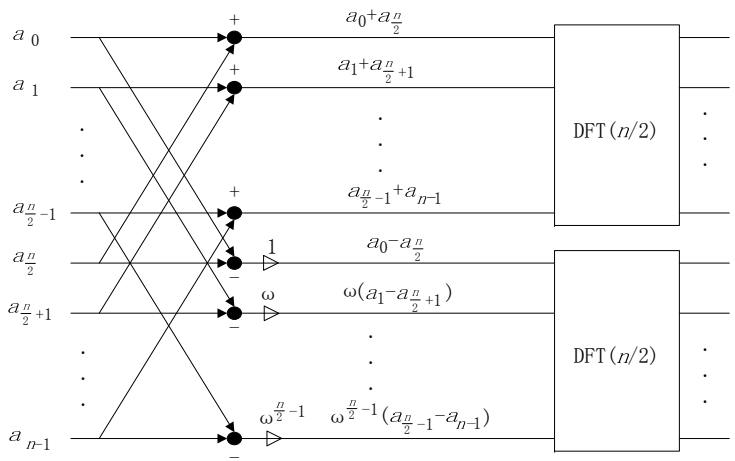
奇数时：$b_l'' = b_{2l+1} = \sum_{k=0}^{n-1} \omega^{(2l+1)k} a_k$

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

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$DFT$



FFT的蝶式递归计算图



n=8的FFT蝶式计算图

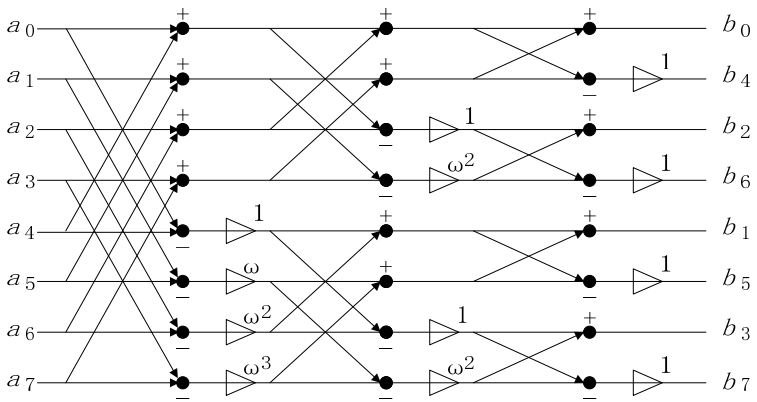


图11.4

蝶式计算示例

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

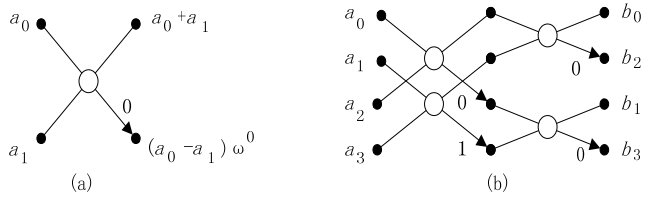
当$n=4$时，$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 1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

对调b_1和b_2，$\Rightarrow$

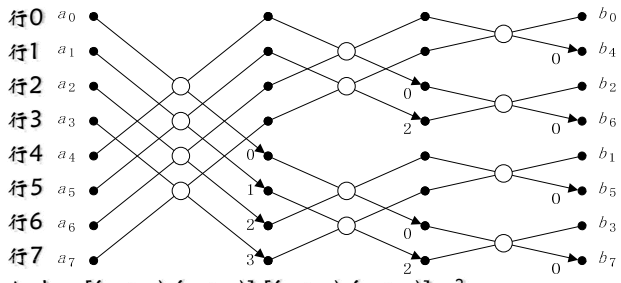
$$\begin{bmatrix} b_0 \\ b_2 \\ b_1 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & \omega & 0 & -\omega \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3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

串行FFT迭代算法

- 蝶式计算流图



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迭代算法

- 算法11.1(P266): 输入: a序列; 输出b序列
- begin
- (1)for k=0 to n-1 do $c_k=a_k$ endfor
- (2)for h=logn-1 to 0 do
- (2.1)$p=2^h$ // p为当前交叉点个数
- (2.2)q=n/p (2.3)$z=\omega^{q/2}$ // z为有p个交叉点时的单位园根
- (2.4) for k=0 to n-1 do
- if (k mod p = k mod 2p) then
- (i) $c_k=c_k+c_{k+p}$
- (ii)$c_{k+p}=(c_k-c_{k+p})z^{k \bmod p}$
- endif
- endfor
- endfor // r(k)为k的位序反
- (3)for k=0 to n-1 do $b_k=c_{r(k)}$ endfor // $C_{r(k)}$表示位置k处的b
- end

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

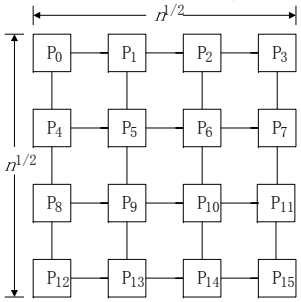


图11.5

SIMD-MC^{2}上的FFT算法

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

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
(2.1)$p=2^h$ (2.2)q=n/p (2.3)$z=\omega^{q/2}$
(2.4)if (k mod p = k mod 2p) then par-do //满足条件的处理器同时做
(i) $c_k=c_k+c_{k+p}$ //(i)和(ii)同时执行
(ii)$c_{k+p}=c_k-c_{k+p}z^{k \bmod p}$
endif
endfor
endfor
(3)for k=0 to n-1 par-do $b_k=c_{r(k)}$ endfor
end

★: n=16
h=3, p=8, q=2, $z=\omega^1$
h=2, p=4, q=4, $z=\omega^2$
h=1, p=2, q=8, $z=\omega^4$
h=0, p=1, q=16, $z=\omega^8$

- 分析:

$$n=2^t$$

$$p=1,2,4,8,\ldots,\frac{n}{2}$$

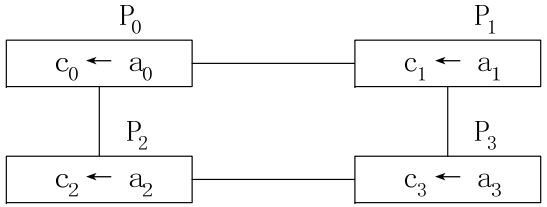
$$s=1,2,4,\ldots,\frac{\sqrt{n}}{2},1,2,4,\ldots,\frac{\sqrt{n}}{2}$$

$$\sum_{k=0}^{\frac{t}{2}-1} s=2(1+2^1+2^2+\ldots+2^{\frac{t}{2}-1})=2(2^{\frac{t}{2}}-1)=2(2\sqrt{n}-1)$$

SIMD-MC^{2}上的FFT算法

- **示例：** P271例11.5, n=4

第(1)步：



SIMD-MC^{2}上的FFT算法

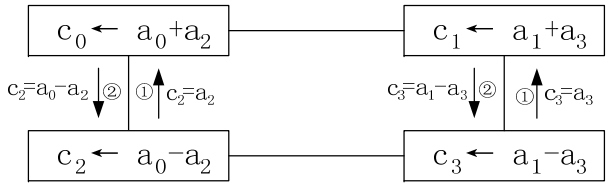
第(2)步：

第1次迭代(h=1): p=2, q=2, z= ω^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: $c_0=c_0+(\omega^2)^0c_2=a_0+a_2$ P_1: $c_1=c_1+(\omega^2)^0c_3=a_1+a_3$

$c_2=c_0-(\omega^2)^0c_2=a_0-a_2$ $c_3=c_1-(\omega^2)^0c_3=a_1-a_3$



SIMD-MC^{2}上的FFT算法

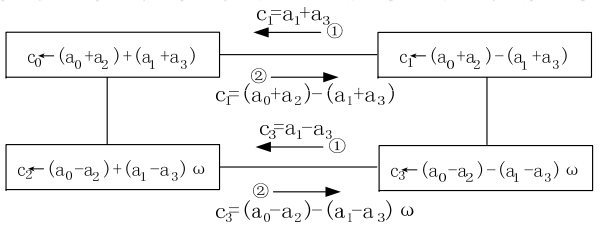
第(2)步：

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

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



SIMD-MC^{2}上的FFT算法

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: $t_{comp}=O(\log n)$
  - 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$t(n)=O(n^{1/2})$

第二次作业题

说明：

1. 任选一题，完成后写出报告，电子邮件方式提交
2. 截至时间本学期第17周；

1. 写出2D环绕上的长方矩阵乘法的算法，写明算法思想和典型的可选方案，并进行复杂度分析。
2. 写出一个在SIMD超立方体模型上的FFT算法，写明算法思想和典型的可选方案，并进行复杂度分析。
3. 应用PCAM方法学设计一个并行应用（选择一个计算生物学问题、有限元应用问题、或者工程模拟问题等）。
4. 写出一个典型的博弈树搜索问题（或其他同等问题）的并行算法，给出基于BSP模型和LogP模型的算法描述，进行算法分析。
5. 写出一个典型的数据挖掘问题的并行算法，给出基于BSP模型和LogP模型的算法描述，进行算法分析。
6. 编程实现求凸多边形的并行算法，对运行结果进行演示（选区若干运行界面并加以说明）
7. 写出处理器数小于n^3的n×n矩阵DNS算法，进行算法分析。