PRAM算法

汪小林

北京大学计算机系

主要内容

- 加速比与**Amdahl**法则
- 并行计算模型
- 基本技术和算法
  - 前缀计算、表排序、选择、归并、排序

加速比与Amdahl法则

- 并行的目标是缩短问题处理的时间
- 对于问题 π
  - 设$S(n)$为最好的顺序算法的(渐进)运行时间
  - 设$T(n, p)$为在p个处理器上并行算法的(渐进)运行时间
  - 则该并行算法的(渐进)**加速比(speedup)**为 $\dfrac{S(n)}{T(n,p)}$
  - 如果 $\dfrac{S(n)}{T(n,p)} = \Theta(p)$，则并行算法具有**线性加速比**

加速比与Amdahl法则

- $pT(n, p)$表示并行算法的**总任务完成量**
- 则并行算法的**效率**为 $\dfrac{S(n)}{pT(n,p)}$
- 当$pT(n, p)=O(S(n))$时，称并行算法是**工作最优**的
- 并行算法是最优的 <=> 并行算法具有线性加速比
- 思考
  - 理论上，并行算法的加速比能否大于 p ?
  - 在实际应用中呢?

加速比与Amdahl法则

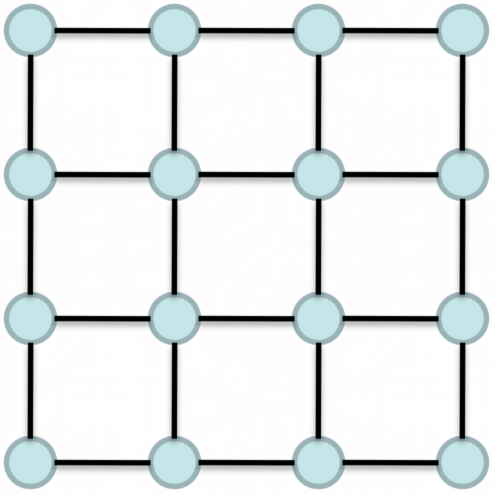
- 怎样确定一个问题的最大并行加速比？
- **Amdahl法则**
  - 以问题的全部任务工作量为 **1**
  - 其中无法并行的工作量为 f
  - 所使用的处理器数量为 p
  - 则最大加速比为 $$\frac{1}{f+\frac{1-f}{p}}$$

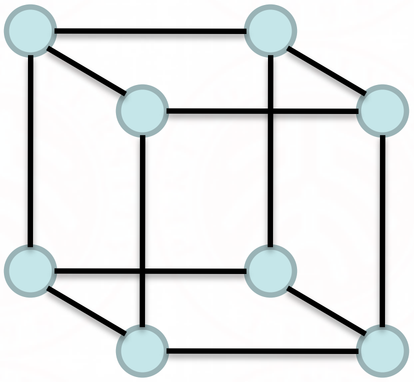
并行计算模型

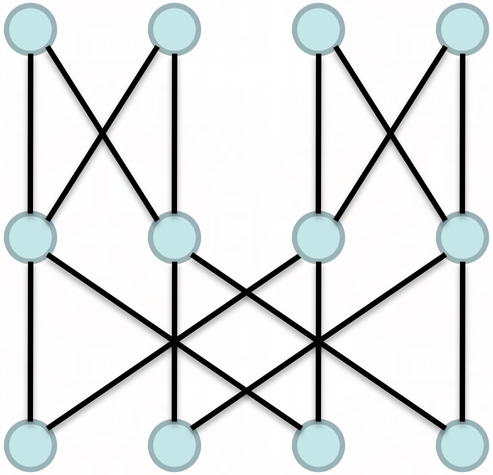
- **RAM**——随机存取机
  - 被广泛接受的顺序计算模型
  - 单位时间内完成各种基本操作：
    加、减、乘、除、比较、存储器访问和赋值等
- 并行计算需要并行处理器间通信
  - 每个处理器可以看做一个**RAM**
  - 通过通信协调各自处理的子任务
  - 通过通信了解已经完成的子任务（在哪些处理器上）
  - 不同的通信方式区分了不同的并行模型
    - 固定连接模型、共享存储模型

并行计算模型

- 固定连接模型

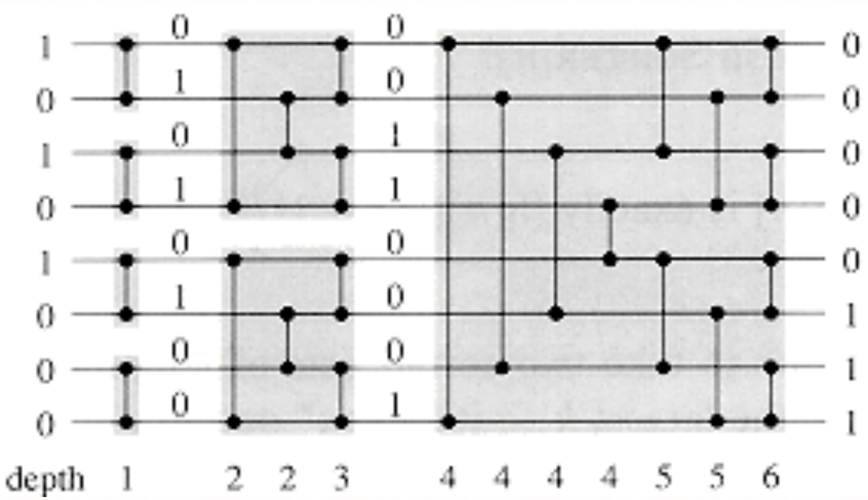

网格结构


超立方体结构


蝶形结构

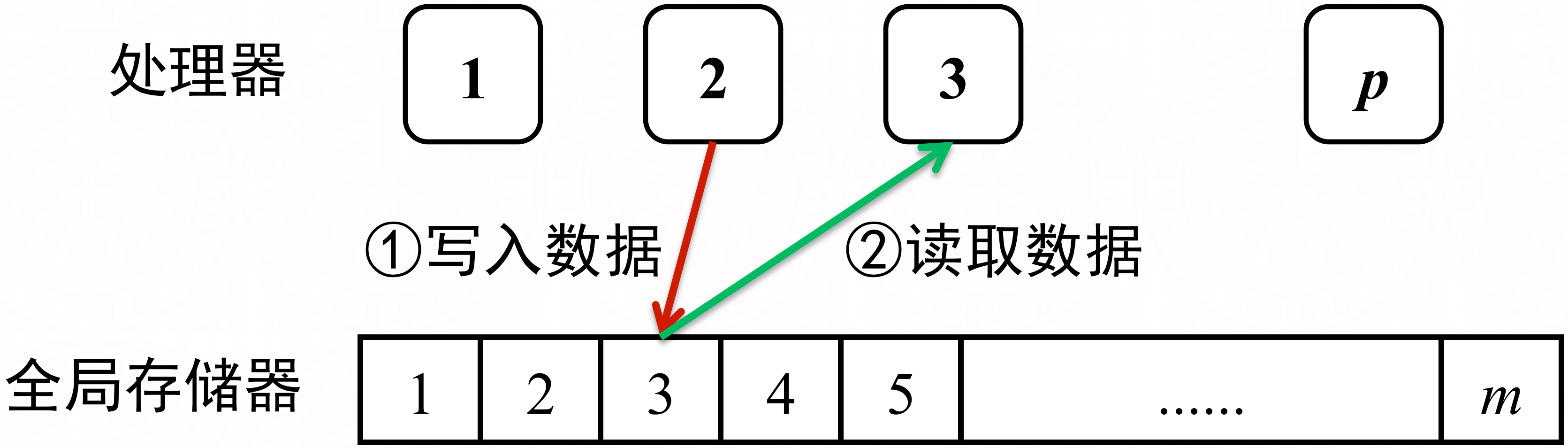
并行计算模型

- 排序网络模型



并行计算模型

- **PRAM**——并行随机存取机



PRAM模型的变体

- 冲突：当不同的处理器同时访问同一个存储器时
- 几种解决冲突的方法形成不同的**PRAM**模型
  - **EREW**——独占式读写
    不允许同时读/写相同的存储器单元
    不同的处理器可以同时读写不同的存储器单元
  - **CREW**——并行读/独占写
    可以并行的读，但不允许并行的写
  - **CRCW**——并行读/并行写
    允许并行的读，也允许并行的写
    Common CRCW、**Arbitraray CRCW**、**Priority CRCW**

各种PRAM的计算能力

- 三种CRCW PRAM
  - Common CRCW——公共并行读写
    - 当要写到相同单元的内容相同时，允许并行写
  - Arbitraray CRCW——任意并行读写
    - 当并行写相同单元时，未知的某一个会成功
  - Priority CRCW——优先并行读写
    - 每个处理器有静态优先级，并行写时高优先级写成功
- 计算能力关系

EREW(p, $T(n, p)$) ⊂ CREW(p, $T(n, p)$) ⊂ Common CRCW(p, $T(n, p)$) ⊂ Arbitraray CRCW(p, $T(n, p)$) ⊂ Priority CRCW(p, $T(n, p)$)

CRCW PRAM上并行算法例子

- 布尔或问题：$A[0] = A[1] \mid A[2] \mid \ldots \mid A[n]$
  - 在RAM上的顺序算法运行时间为$O(n)$
  - 在CRCW PRAM上$O(1)$时间的并行算法

程序13.1 在$O(1)$时间内计算布尔或

```
Processor i (in parallel for 1 <= i <= n) do
        if (A[i]) A[0] = A[i];
```

  - 程序13.1能在三种CRCW PRAM上运行
  - 不能在EREW或CREW PRAM上运行
  - 布尔或问题在EREW或CREW PRAM上的并行复杂度为$O(\log n)$

PRAM上的减慢引理

- 减慢引理
  - 在p处理器计算机上运行时间为T的任何并行算法可以在$p'(p'<p)$处理器计算机上运行，其运行时间为$O(pT/p')$
- 程序13.1在p处理器上的运行时间
  - 使用n个处理器，运行时间为$O(1)$
  - 在$n/\log n$个处理器上，运行时间为$O(\log n)$
  - 在 $\sqrt{n}$个处理器上，运行时间为$O(\sqrt{n})$
  - 当$p=1$时，算法运行时间为$O(n)$，与最佳顺序算法的运行时间相同

基本技术和算法

- 前缀计算问题
- 链表排序问题
- 选择问题
- 归并问题
- 排序问题

前缀计算问题

- 设+是域Σ上的二元结合运算符，即满足 $((x+y)+z)=(x+(y+z)), x, y, z \in \Sigma$
- 前缀计算问题
  – 对于$x_1, x_2, x_3, ..., x_n \in \Sigma$，计算$n$个元素$x_1, x_1+x_2, x_1+x_2+x_3, ..., x_1+x_2+...+x_n$，计算结果就是前缀
- 例子1：整数上的加法运算
  – 输入3,-5,8,2,5,4；前缀计算输出3,-2,6,8,13,17
- 例子2：整数上的最小值运算
  – 输入5,8,-2,7,-11,12；前缀计算输出5,5,-2,-2,-11,-11

$O(\log n)$时间实现前缀计算

程序13.2 使用$O(\log n)$时间实现前缀计算

第0步 如果$n==1$，则一个处理器输出x_1。

第1步 让前$n/2$个处理器递归的计算$x_1, x_2, \ldots, x_{n/2}$的前缀，假设结果为$y_1, y_2, \ldots, y_{n/2}$。同时，让其余的处理器递归地计算$x_{n/2+1}, x_{n/2+2}, \ldots, x_n$的前缀，假设结果为$y_{n/2+1}, y_{n/2+2}, \ldots, y_n$。

第2步 注意最终结果的前一半与$y_1, y_2, \ldots, y_{n/2}$相同，后一半是$y_{n/2}+y_{n/2+1}, y_{n/2}+y_{n/2+2}, \ldots, y_{n/2}+y_n$。让后$n/2$个处理器并行的从全局存储器中读取$y_{n/2}$并更新。这一步需要的时间为$O(1)$。

$O(\lg n)$时间实现前缀计算

输入

x_1	x_2	……	x_n

分治

x_1	x_2	……	$x_{n/2}$

y_1	y_2	……	$y_{n/2}$

前$n/2$个处理器
递归计算前缀

后$n/2$个处理器
递归计算前缀

$x_{n/2+1}$	$x_{n/2+2}$	……	x_n

$y_{n/2+1}$	$y_{n/2+2}$	……	y_n

$O(\lg n)$时间实现前缀计算

前$n/2$个处理器直接输出前半部分结果

y_1	y_2	……	$y_{n/2}$

后$n/2$个处理器计算输出后半部分结果

$y_{n/2}+y_{n/2+1}$	$y_{n/2}+y_{n/2+2}$	……	$y_{n/2}+y_n$

$O(\log n)$时间实现前缀计算 – 分析

运行时间： $T(n, n)=T(\mathrm{n}/2,\mathrm{n}/2)+O(1)=O(\log n)$

加速比： $\dfrac{S(n)}{T(n,n)}=O\left(\dfrac{n}{\log n}\right)$ $S(n)=O(n)$

效率： $\dfrac{S(n)}{nT(n,n)}=O\left(\dfrac{1}{\log n}\right)$

工作最优的对数时间前缀计算算法

程序13.3 工作最优的对数时间前缀计算算法

第1步 处理器i (i=1, 2, …, $n/\log n$) 并行的计算$\log n$个分配给它的元素$x_{(i-1)\log n+1}, x_{(i-1)\log n+2}, \ldots, x_{i\log n}$的前缀。所用时间为$O(\log n)$。令结果为$z_{(i-1)\log n+1}, z_{(i-1)\log n+2}, \ldots, z_{i\log n}$。

第2步 所有$n/\log n$个处理器共同利用程序13.2计算$n/\log n$个元素的前缀$z_{\log n}, z_{2\log n}, z_{3\log n}, \ldots, z_n$。令结果为$w_{\log n}, w_{2\log n}, w_{3\log n}, \ldots, w_n$。

第3步 每一个处理器按如下方法更新在第1步计算的前缀：对于i=2,3,…, $n/\log n$，处理器i计算并输出$w_{(i-1)\log n}+w_{(i-1)\log n+1}, w_{(i-1)\log n}+w_{(i-1)\log n+2}, \ldots, w_{(i-1)\log n}+w_{i\log n}$，处理器1直接输出$z_1, z_2, \ldots, z_{\log n}$。

工作最优的对数时间前缀计算算法

1) $p=n/\log n$，$i=1, 2, \ldots, n/\log n$，每个处理器i计算$\log n$个元素

$x_{(i-1)\log n+1}$	$x_{(i-1)\log n+2}$	……	$x_{i\log n}$

$z_{(i-1)\log n+1}$	$z_{(i-1)\log n+2}$	……	$z_{i\log n}$

2) 用$n/\log n$个处理器计算$n/\log n$个元素的前缀，时间$O(\log n)$

$z_{\log n}$	$z_{2\log n}$	……	$z_{(n/\log n)\log n}$

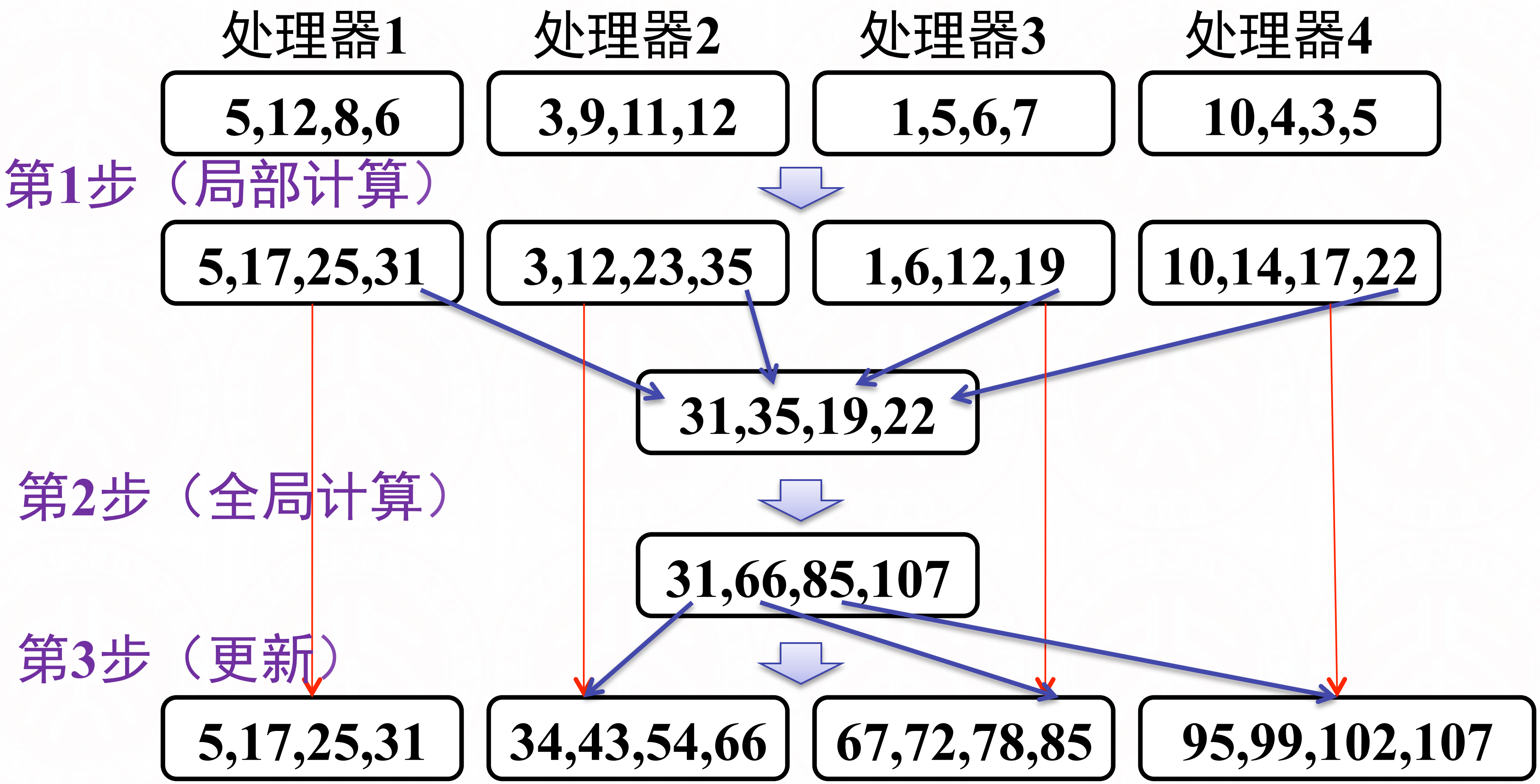
$w_{\log n}$	$w_{2\log n}$	……	w_n

工作最优的对数时间前缀计算算法

3) $i = 2, \ldots, n/\log n$，每个处理器i计算$\log n$个元素的完整前缀

$w_{(i-1)\log n+1}$ $z_{(i-1)\log n+1}$	$w_{(i-1)\log n+2}$ $z_{(i-1)\log n+2}$	……	$w_{(i-1)\log n+i\log n}$ $z_{i\log n}$

工作最优前缀计算算法的例子



$O(\lg n)$时间实现前缀计算 – 分析

运行时间： $T(n, n/\log n)=O(\log n)+O(\log n)+O(\log n)=O(\log n)$

加速比： $$\frac{S(n)}{T(n,n/\log n)}=O\left(\frac{n}{\log n}\right)$$ $S(n)=O(n)$

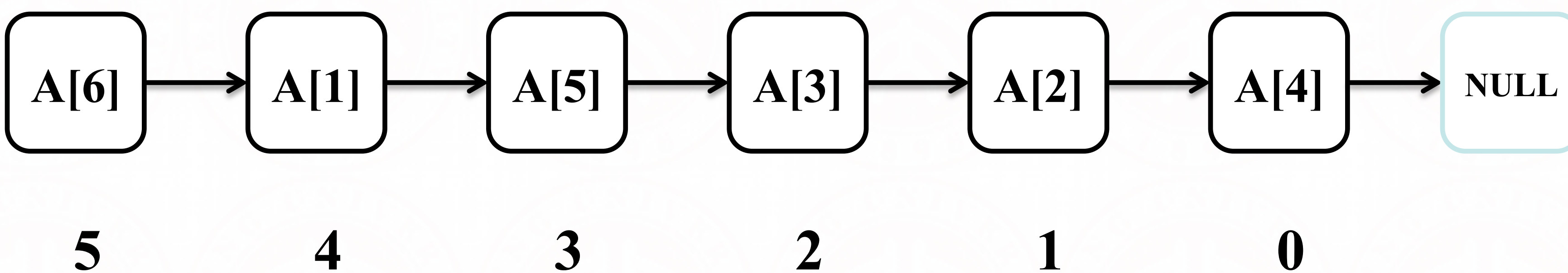
效率： $$\frac{S(n)}{(n/\log n)T(n,n/\log n)}=O(1)$$

链表排序问题

- 链表排序问题
  - 对于单链表，计算每个节点后面的节点数
  - 简化：用数组$A[1..n]$表示单链表，$A[i]=j$
- 链表排序问题的顺序算法
  - 首先遍历数组$A[1..n]$找到唯一未出现的数 i
  - $A[i]$是链表的头节点
  - 从$A[i]$其顺序遍历链表，每个节点后面的节点数依次为 **$(n-1), (n-2), \dots, 0$**

链表排序问题

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
5	4	2	0	3	1



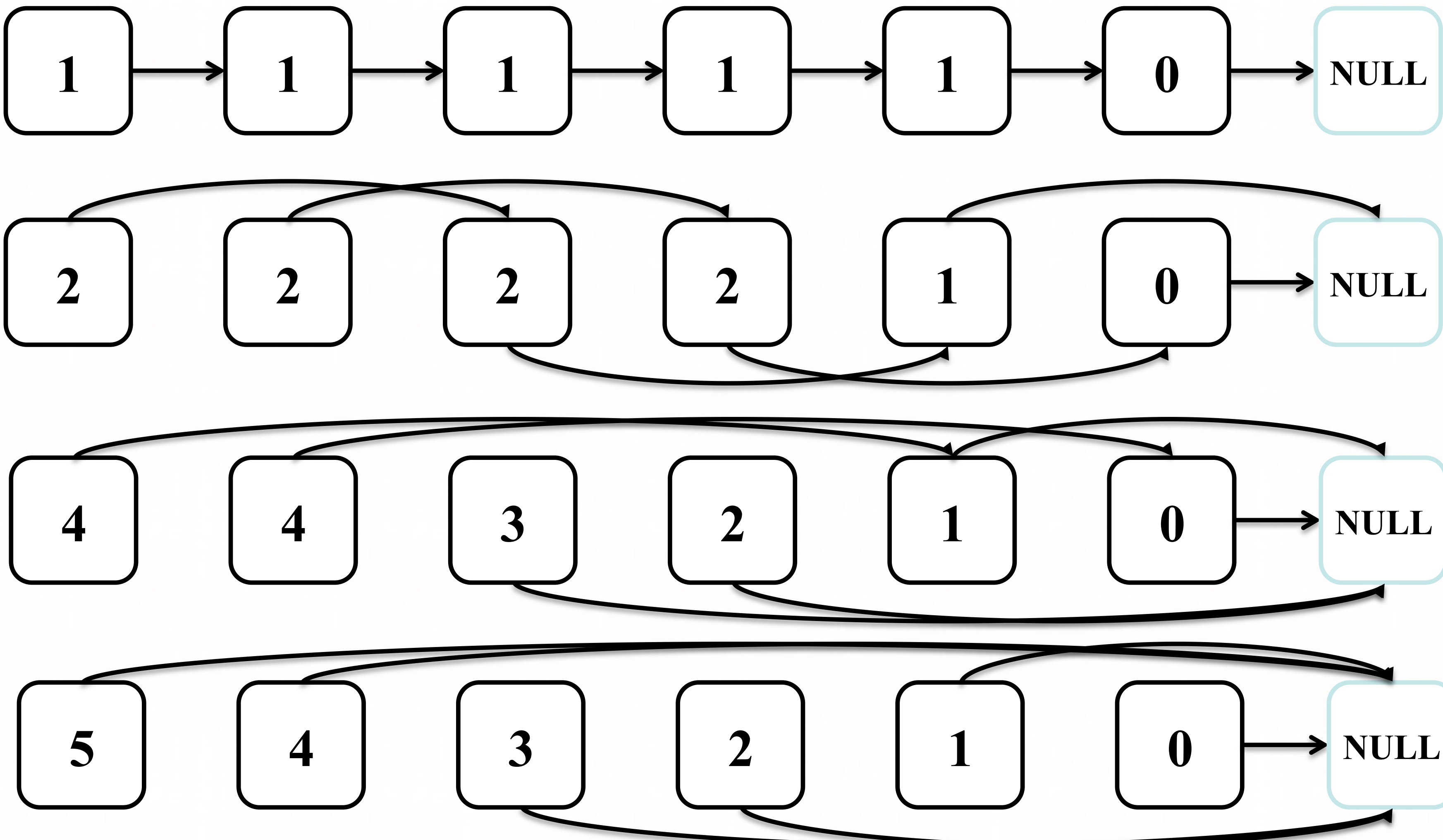
$O(\log n)$的链表排序算法

程序13.4 运行时间为$O(\log n)$的链表排序算法

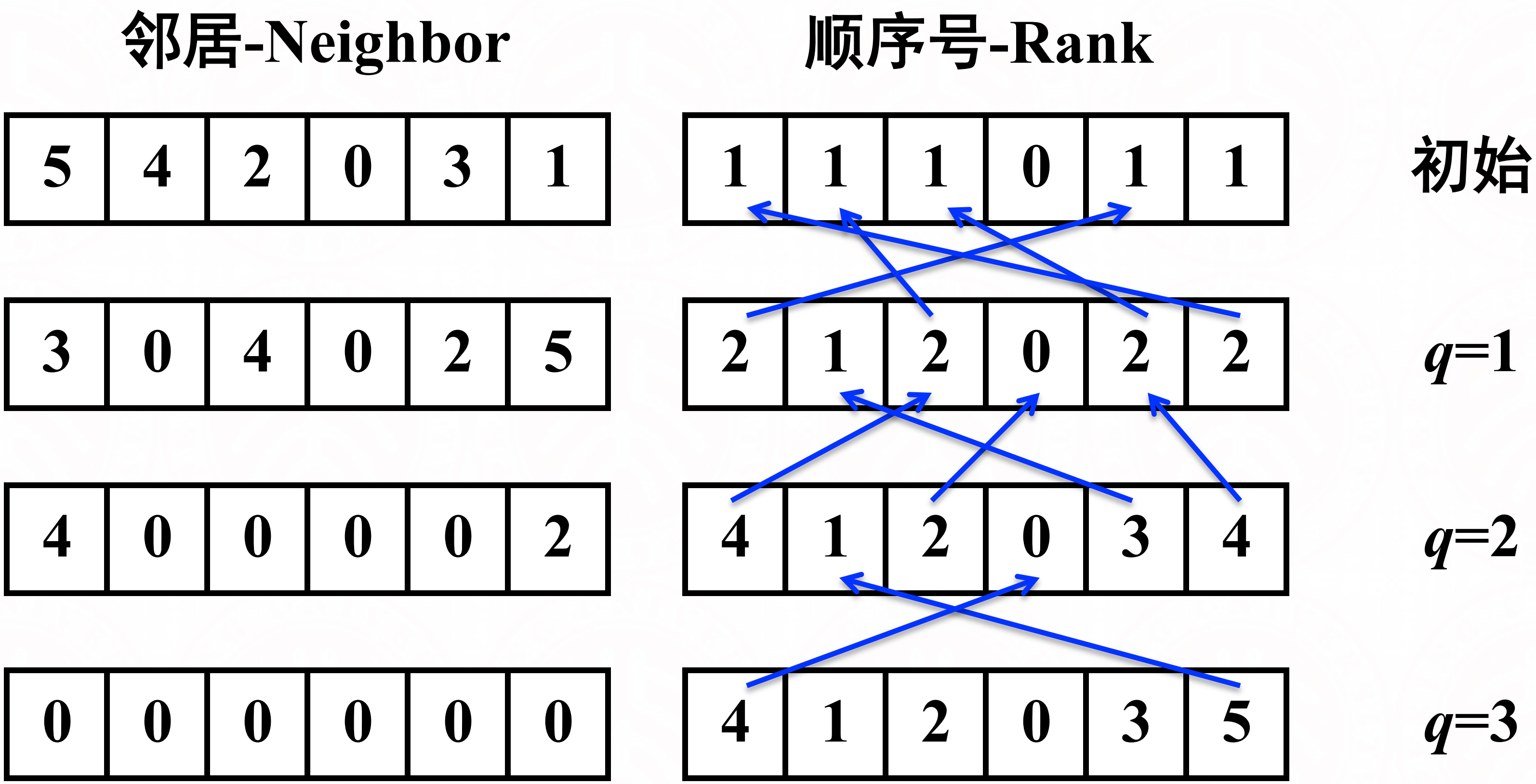
首先设置每个节点的$Rank[i]=1$，链表尾节点$Rank[]=0$

```
for (int q=1; q<=⌈logn⌉; q++)
    Processor i (in parallel for 1 <= i <= n) do:
        if (Neighbor[i] != 0) {
                Rank(i) += Rank[Neighbor[i]];
                Neighbor[i] = Neighbor[Neighbor[i]];
        }
```

$O(\log n)$的链表排序算法



$O(\log n)$的链表排序算法的例子



$O(\log n)$的链表排序算法 – 分析

运行时间： $T(n, n)=O(\log n)$

加速比： $$\frac{S(n)}{T(n,n)}=O\left(\frac{n}{\log n}\right)$$ $S(n)=O(n)$

效率： $$\frac{S(n)}{(n)T(n,n)}=O\left(\frac{1}{\log n}\right)$$

工作最优的随机链表排序算法

程序13.5 工作最优的随机链表排序算法

步骤0 $n/\log n$个处理器并行的计算获得双链表($Rank[x]$, $L[x]$, $R[x]$)

while (剩余的节点数目>2) {

步骤1 处理器 i 考查某个节点 x，以1/2概率标记要编出节点 x

步骤2 处理器 i 检查如果$R[x]$未被标记，则确定编出 x，否则清除标记

步骤3 处理器 i 编出x，在 x 中记录编出阶段号、$L[x]$和$Rank[L[x]]$，

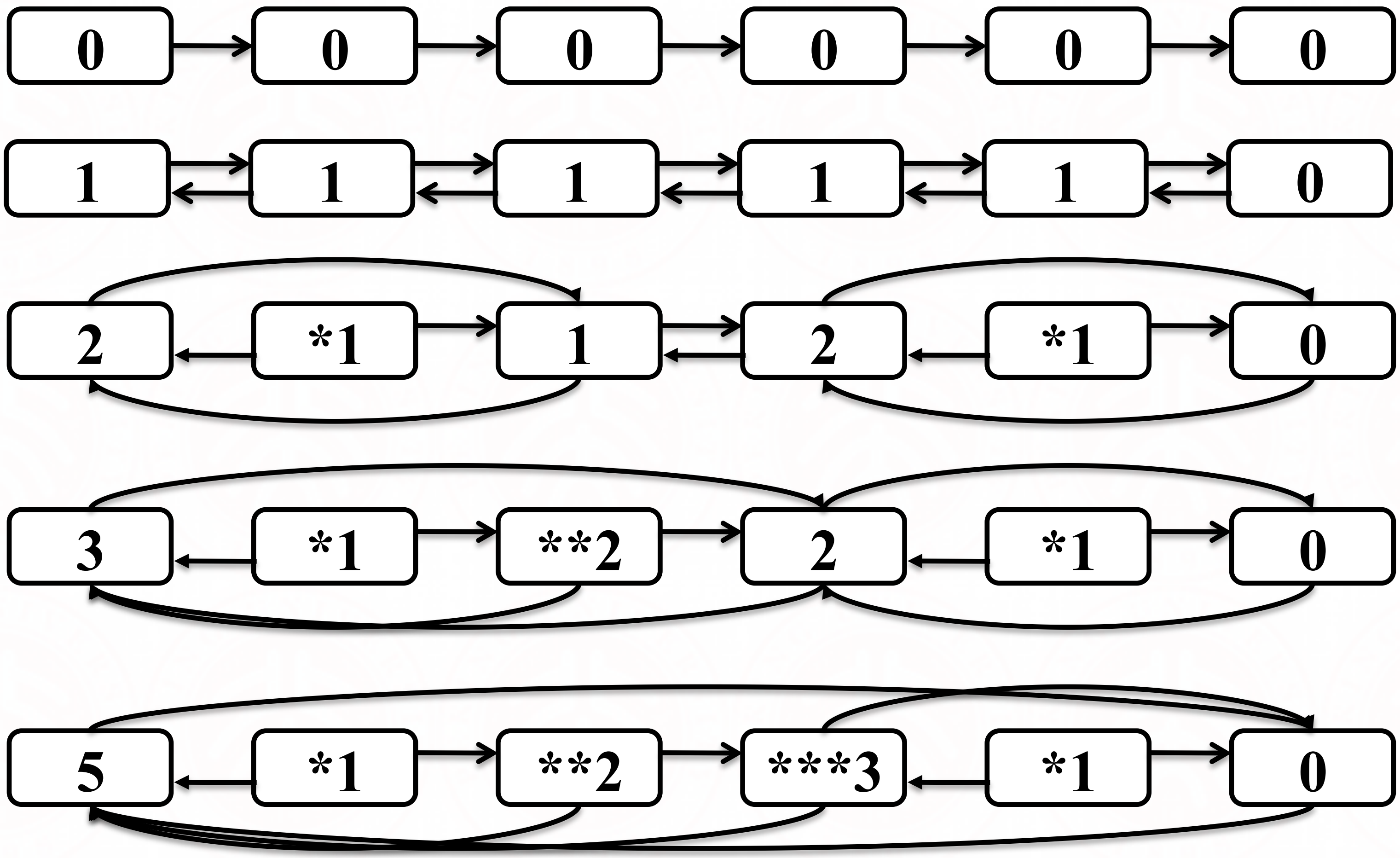
$Rank[L[x]]$ +=$Rank[x]$， $L[R[x]]$ =$L[x]$，$R[L[x]]$=$R[x]$；

}

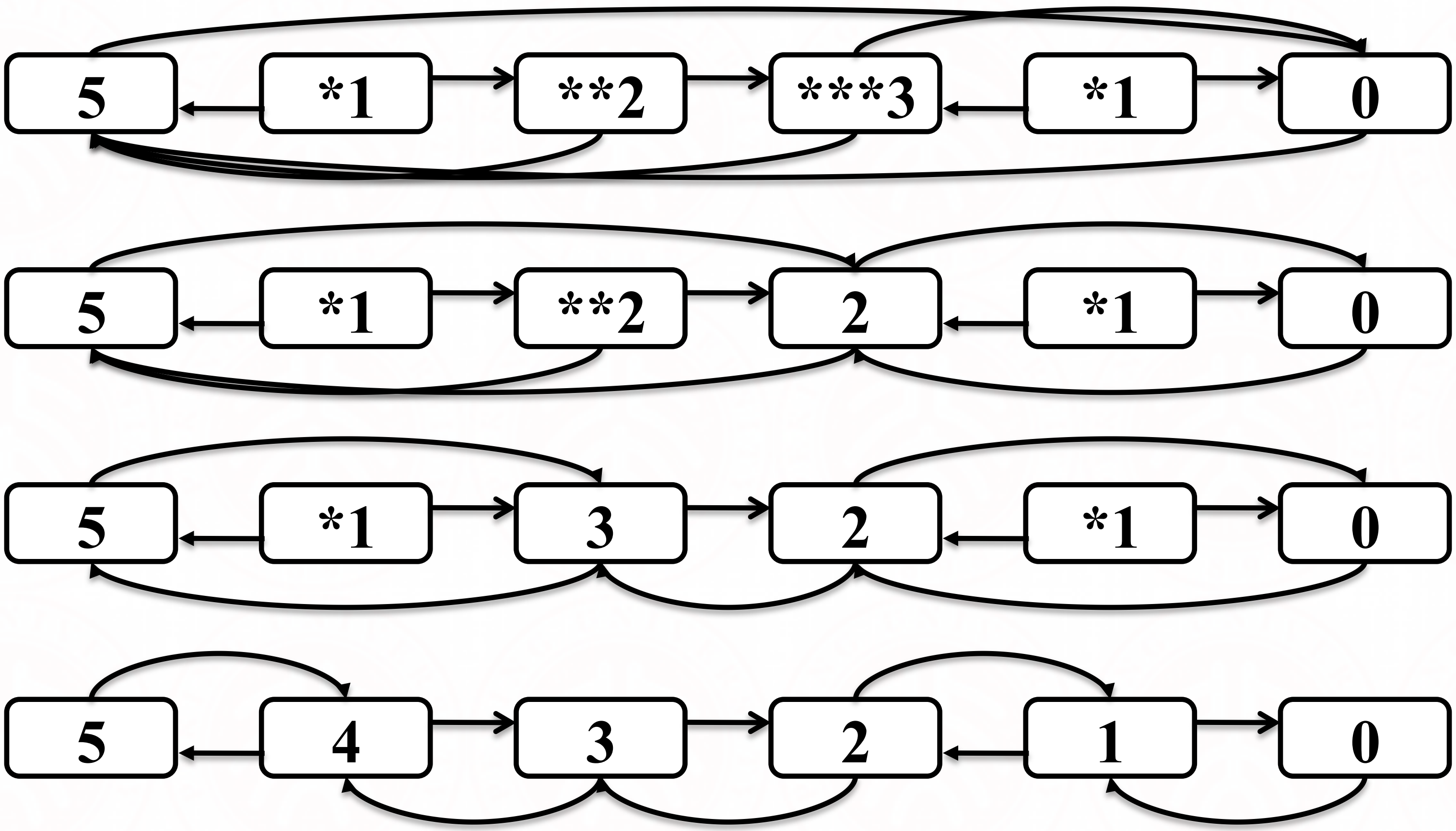
步骤4 反向依次重新编入被编出节点 x，$Rank[x]$=$Rank[L[x]]$-$Rank[x]$，

$R[L[x]]$=x ，$L[R[x]]$=x

工作最优的随机链表排序算法例子



工作最优的随机链表排序算法例子



工作最优的随机链表排序算法 - 分析

运行时间： $T(n, n/\log n)=O(1)+O(\log n)+O(\log n)=O(\log n)$

加速比： $$\frac{S(n)}{T(n,n/\log n)}=O\left(\frac{n}{\log n}\right)$$ $S(n)=O(n)$

效率： $$\frac{S(n)}{(n/\log n)T(n,n/\log n)}=O(1)$$

选择问题 – O(1)时间的最大值算法

程序13.6 在$O(1)$时间内寻找最大值

第0步 如果$n==1$，则输出关键字。

第1步 处理器p_{ij}（对于每个$1<=i, j<=n$）并行的计算$x_{ij}=(k_i<k_j)$。

第2步 n^2个处理器被分为n个组$G_1, G_2, \ldots, G_n$，这里G_i（$1<=i<=n$）由处理器$p_{i1}, p_{i2}, \ldots, p_{in}$组成，每一个组$G_i$计算$x_{i1}, x_{i2}, \ldots, x_{in}$的布尔或。

第3步 如果G_i在第2步计算出一个0，那么处理器p_{i1}输出k_i作为结果。

O(1)时间的最大值算法 - 分析

运行时间： $T(n, n^2)=O(1)+O(1)+O(1)=O(1)$

加速比： $$\frac{S(n)}{T(n,n^2)}=O(n)$$ $S(n)=O(n)$

效率： $$\frac{S(n)}{n^2T(n,n^2)}=O\left(\frac{1}{n}\right)$$

$O(\log\log n)$时间的最大值算法

程序13.7 在$O(\log\log n)$时间内寻找最大值

第0步 如果$n==1$，则返回k_1。

第1步 将输出分成$\sqrt{n}$个部分$K_1, K_2, \ldots, K_{\sqrt{n}}$，$K_i$由$k_{(i-1)\sqrt{n}+1}$，$k_{(i-1)\sqrt{n}+2}$，…，$k_{i\sqrt{n}}$组成，类似的，也将处理器分成几个部分，以使$P_i$（$1<=i<=n$）由处理器$p_{(i-1)\sqrt{n}+1}$，$p_{(i-1)\sqrt{n}+2}$，…，$p_{i\sqrt{n}}$组成。由$P_i$递归的寻找$K_i$的最大值，其中$1<=i<=n$。

第2步 如果$M_1, M_2, \ldots, M_n$为各部分的最大值，使用程序13.6寻找并输出它们的最大值。

O(loglog*n*)时间的最大值算法 - 分析

运行时间： $T(n, n)=T(\sqrt{n},\sqrt{n}) + O(1) = O(\log\log n)$

加速比： $\dfrac{S(n)}{T(n,n)} = O\left(\dfrac{n}{\log\log n}\right)$ $S(n)=O(n)$

效率： $\dfrac{S(n)}{nT(n,n)} = O\left(\dfrac{1}{\log\log n}\right)$

工作最优的随机选择算法

程序13.9 一个工作最优的随机选择算法

$N:=n$; // N在任何时间都是有效关键字的数目

While ($N > n^{0.4}$) {

第1步 每一个有效关键字以概率$1/N^{\varepsilon}$位于随机样本S中，这一步需要$\log n$时间，$O(N^{1-\varepsilon})$个关键字（来自所有的处理器）以高概率位于随机样本中。

第2步 所有的处理器执行一次前缀和运算，计数样本中关键字数目q。将q广播到所有的处理器。如果q不在$[0.5 N^{1-\varepsilon}, 1.5 N^{1-\varepsilon}]$区间内，则转到第1步。

第3步 集中并排序样本关键字。

工作最优的随机选择算法

第4步 从S中选择顺序号分别为$iq/N-d\sqrt{(q\log N)}$和$iq/N+d\sqrt{(q\log N)}$的关键字l_1和l_2，其中d是一个大于3α的常数。将l_1和l_2广播到所有的处理器，待选择的关键字的值会以高概率位于$[l_1, l_2]$内。

第5步 计算$[l_1, l_2]$区间内有效关键字的数目r，还要计算$< l_1$的有效关键字的数目t。将r和t广播到所有的处理器。如果i不在$(t,t+r]$区间内或者$r!=O(N^{(1+\epsilon)/2}\sqrt{\log N})$，则转到第1步；否则，删掉所有值小于$l_1$或大于$l_2$的所有关键字，并设置$i:=i-t$, $N:=r$。

}

第6步 集中并排序有效关键字，确定并输出第i小的关键字。

归并问题

- 对数时间算法
  - $X_1=k_1, k_2, k_3, \ldots, k_m$和$X_2=k_{m+1}, k_{m+2}, \ldots, k_{2m}$
  - 假设所有的关键字互不相同
  - 可用$n=2m$个处理器
  - 每个处理器二分法求$X_1 \cup X_2$中一个关键字 k 的顺序号
  - 把顺序号为 i 的关键字写入第 i 个存储单元
- 分析
  - 运行时间：$O(\log m)$
  - 加速：$O(m/\log m)$
  - 效率：$O(1/\log m)$

奇偶归并算法

程序13.10 奇偶归并算法

第0步 如果m==1，用一次比较归并序列。

第1步 将X_1和X_2划分成奇偶两部分。

即将X_1分为$X_1^{odd} = k_1, k_3, \ldots, k_{m-1}$和$X_1^{even} = k_2, k_4, \ldots, k_m$。

类似地，将X_2分为X_2^{odd}和X_2^{even}。

第2步 利用m个处理器递归地归并X_1^{odd}和X_2^{odd}，设结果为$L_1= l_1, l_2, \ldots, l_m$。

注意X_1^{odd}、X_1^{even}、X_2^{odd}和X_2^{even}是有序的。

同时，用另外的m个处理器归并X_1^{even}和X_2^{even}得到$L_2= l_{m+1}, l_{m+2}, \ldots, l_{2m}$。

奇偶归并算法

第3步 将L_1和L_2洗牌，即形成序列$l_1, l_{m+1}, l_2, l_{m+2}, \ldots, l_m, l_{2m}$。比较每一个序对$(l_{m+i}, l_{i+1})$中的元素，如果顺序不对，则进行交换。也就是，比较$l_{m+1}$和$l_2$，如果需要则交换；比较$l_{m+2}$和$l_3$，如果需要则交换；如此类推。最后输出结果顺序。

奇偶归并算法

$X_1 =$ 2, 5, 8, 11, 13, 16, 21, 25

X_1^{odd}: 2, 8, 13, 21

X_1^{even}: 5, 11, 16, 25

$X_2 =$ 4, 9, 12, 18, 23, 27, 31, 34

X_2^{odd}: 4, 12, 23, 31

X_2^{even}: 9, 18, 27, 34

归并

$L_1 =$ 2, 4, 8, 12, 13, 21, 23, 31

归并

$L_2 =$ 5, 9, 11, 16, 18, 25, 27, 34

洗牌

$L =$ 2, $\boxed{5, 4}$, $\boxed{9, 8}$, $\boxed{11, 12}$, $\boxed{16, 13}$, $\boxed{18, 21}$, $\boxed{25, 23}$, $\boxed{27, 31}$, 34

⇓ 比较－交换

2, 4, 5, 8, 9, 11, 12, 13, 16, 18, 21, 23, 25, 27, 31, 34

奇偶归并算法 - 分析

运行时间： $T(m, 2m)=T(m/2,m) + O(1) = O(\log n)$

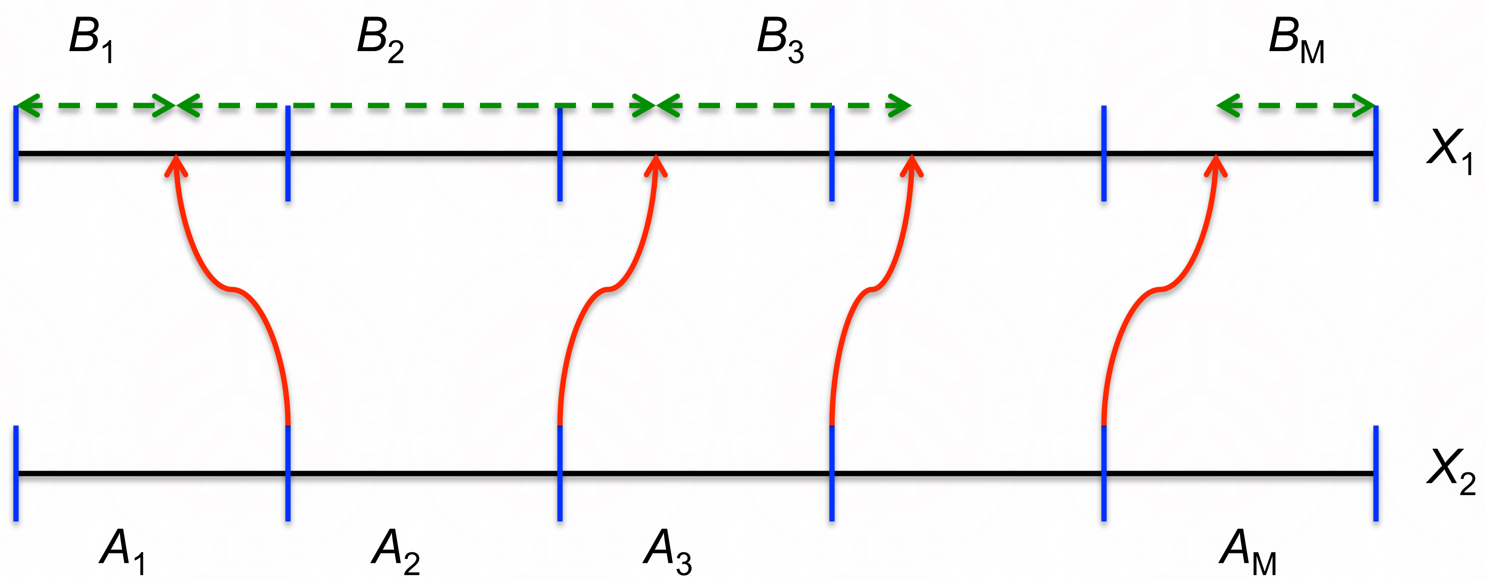
加速比： $$\frac{S(m)}{T(m,2m)} = O\left(\frac{m}{\log m}\right)$$ $S(m)=O(m)$

效率： $$\frac{S(m)}{2mT(m,2m)} = O\left(\frac{1}{\log m}\right)$$

工作最优的归并算法

- 用$m/\log m$个处理器
  1. 先把X_1均分为$A_1, A_2, A_3, ..., A_M$和$M=m/\log m$
  2. 找每个A_i中的最大关键字 l_i 在X_2中的位置，把X_2划分为$B_1, B_2, B_3, ..., B_M$
  3. 对于每对A_i和B_i，如果$|B_i|$较大，对之划分$\lceil |B_i|/\log m \rceil$部分并划分$A_i$
  4. 对于每对区间，用一个处理器在$O(\log m)$时间内归并

工作最优的归并算法



工作最优的归并算法

运行时间： $T(m, 2m/\log m) = O(\log m) + O(\log m) = O(\log n)$

加速比： $$\frac{S(m)}{T(m, 2m/\log m)} = O\left(\frac{m}{\log m}\right)$$ $S(m) = O(m)$

效率： $$\frac{S(m)}{(2m/\log m)T(m, 2m/\log m)} = O(1)$$

O(log log *m*)时间归并算法

程序13.11 在O(log log m)时间内实现归并

第1步

将X_1划分$\sqrt{m}$成个部分，每个部分有$\sqrt{m}$个元素。

称这些部分为$A_1, A_2, \ldots, A_{\sqrt{m}}$。令$A_i$中最大的关键字为$l_i$ (i=1, 2, …, $\sqrt{m}$)。

给每个l_i分配个$\sqrt{m}$处理器，与l_i关联的处理器在X_2上执行$\sqrt{m}$叉查找。在$O(1)$时间内从X_2中找到l_i的正确位置（即已排序的）。这会将X_2划分成$\sqrt{m}$成个部分，注意，某些部分可能是空的。

令的相应部分为$B_1, B_2, \ldots, B_{\sqrt{m}}$，子集$B_i$是$A_i$在$X_2$中的相应子集。

O(log log *m*)时间归并算法

第2步

现在，X_1和X_2的归并简化为A_i和B_i的归并，其中i=1, 2, …, $\sqrt{m}$。

已知每个A_i的大小为$\sqrt{m}$，而B_i的大小可能很大（或者很小）。为了归并A_i和B_i，可以使用多次划分的思想。令A_i和B_i为任一对，

如果$|B_i|=O(\sqrt{m})$，可以用$\sqrt{m}$ 个处理器处理递归归并A_i和B_i。

如果$|B_i|=\omega(m)$，将B_i划分成$\lceil |B_i|/\sqrt{m} \rceil$个部分，每一部分至多有$\sqrt{m}$个$B_i$的连续关键字。给每一部分分配$\sqrt{m}$个处理器，以便能在$O(1)$的时间内找到该部分在$A_i$中的相应子集。从而$A_i$与$B_i$的归并问题简化为$\lceil |B_i|/\sqrt{m} \rceil$个子问题，每个子问题的长度为$O(\sqrt{m})$的两个序列，即可递归求解。

O(log log *m*)时间归并算法 - 分析

运行时间： $T(m, 2m)=T(\sqrt{m},2\sqrt{m}) + O(1) = O(\log\log m)$

加速比： $$\frac{S(m)}{T(m,2m)} = O\left(\frac{m}{\log\log m}\right)$$ $S(m)=O(m)$

效率： $$\frac{S(m)}{2mT(m,2m)} = O\left(\frac{1}{\log\log m}\right)$$

排序问题

- 对数时间的并行排序算法
  - n^2个处理器
  - $O(\log n)$时间内确定每个元素的顺序号
  - 按顺序号输出关键字
- 分析
  - 运行时间：$T(n,n^2)=O(\log n)$
  - 加速：$S(n)/T(n,n^2)=O(n)$
  - 效率：$S(n)/(n^2T(n,n^2))=O(1/n)$

奇偶归并排序

程序13.12 奇偶归并排序

第0步 如果$n<=1$，返回X。

第1步 令$X = k_1, k_2, \ldots, k_n$为输入，将输入划分成两个部分：

$X'_1=k_1, k_2, \ldots, k_{n/2}$和$X'_2=k_{n/2+1}, k_{n/2+2}, \ldots, k_n$。

第2步 利用$n/2$个处理器对X'_1进行递归排序，结果为X_1。

同时，利用另外$n/2$个处理器对X'_2进行递归排序，结果为X_2。

第3步 利用程序13.10用$n=2m$个处理器归并X_1和X_2。

奇偶归并排序 - 分析

运行时间： $T(n, n)=T(n/2, n/2) + O(\log n) = O(\log^2 n)$

加速比： $$\frac{S(n)}{T(n,n)} = O\left(\frac{n}{\log n}\right)$$ $S(m)=O(m)$

效率： $$\frac{S(n)}{nT(n,n)} = O\left(\frac{1}{\log n}\right)$$

Preparata排序算法

程序13.13 Preparata排序算法

第0步 如果n是一个小的常量，使用任意算法对关键字排序，然后退出。

第1步 将n个关键字划分为$\log n$个部分，每一部分有$n/\log n$个关键字。给每一部分分配个n处理器，按并行方式递归地排序每一部分。令$S_1, S_2, \ldots, S_{\log n}$为排序后的序列。

第2步 并行的归并S_i和S_j，其中$1<=i, j<=\log n$，这可以通过给每一对(i, j)分配$n/\log n$个处理器实现。使用$n/\log n$个处理器，这一步可以根据程序13.11在$O(\log \log n)$时间内完成。该归并步骤还计算出了每一个关键字在S_i中的顺序号。

Preparata排序算法

第3步 分配log$n$个处理器用于计算每一个关键字在原输入中的顺序号。对于每一个关键字，将第2步计算得到的log$n$个顺序号相加，对于所有的关键字，该过程是并行的。这可以使用前缀计算算法（见程序13.3）在$O(\log \log n)$时间内完成。最后，按顺序号输出关键字。

Preparata排序算法 - 分析

运行时间： $T(n, n\log n) = T(n/\log n, n) + O(\log\log n) = O(\log n)$

加速比： $\dfrac{S(n)}{T(n, n\log n)} = O(n)$ $S(m)=O(m)$

效率： $\dfrac{S(n)}{(n\log n)T(n, n\log n)} = O\left(\dfrac{1}{\log n}\right)$

Reischuk排序算法

程序13.14 Reischuk排序算法

第1步 $N=n/(\log^4 n)$个处理器都从给定的输入序列$X = k_1, k_2, \ldots, k_n$中随机地取一个关键字作为样本。

第2步 使用Preparata算法排序在第1步中获得的N个关键字。令$l_1, l_2, \ldots, l_N$为排序后的序列。

第3步 定义$K_1=\{k\in X|k\le l_1\}$；$K_i=\{k\in X|l_{i-1}<k\le l_i\}$，$i=2, 3, \ldots, N$；$K_{N+1}=\{k\in X|k>l_N\}$。按定义将输入$X$划分成$K_i$，这通过以下过程实现：首先，寻找每个关键字所属的部分号（并行的使用二分查找）；然后，根据部分号排序关键字，从而实现划分。

第4步 对于$1<=i<=N+1$，并行地执行；使用Preparata算法排序K_i。

第5步 输出排序后的K_1，排序后的K_2，…，K_{N+1}。

小结

- 加速比与Amdahl法则
- 并行计算模型
- 基本技术和算法
  - 前缀计算、表排序、选择、归并、排序

- 参考书《计算机算法》Ellis Horowitz etc 著 冯博琴 等译 机械工业出报社