高性能计算技术

第六讲 并行算法设计（2）

v ώ

华南理工大学计算机学院

内容概要

- 并行算法的基本设计技术
- 并行算法的一般设计过程

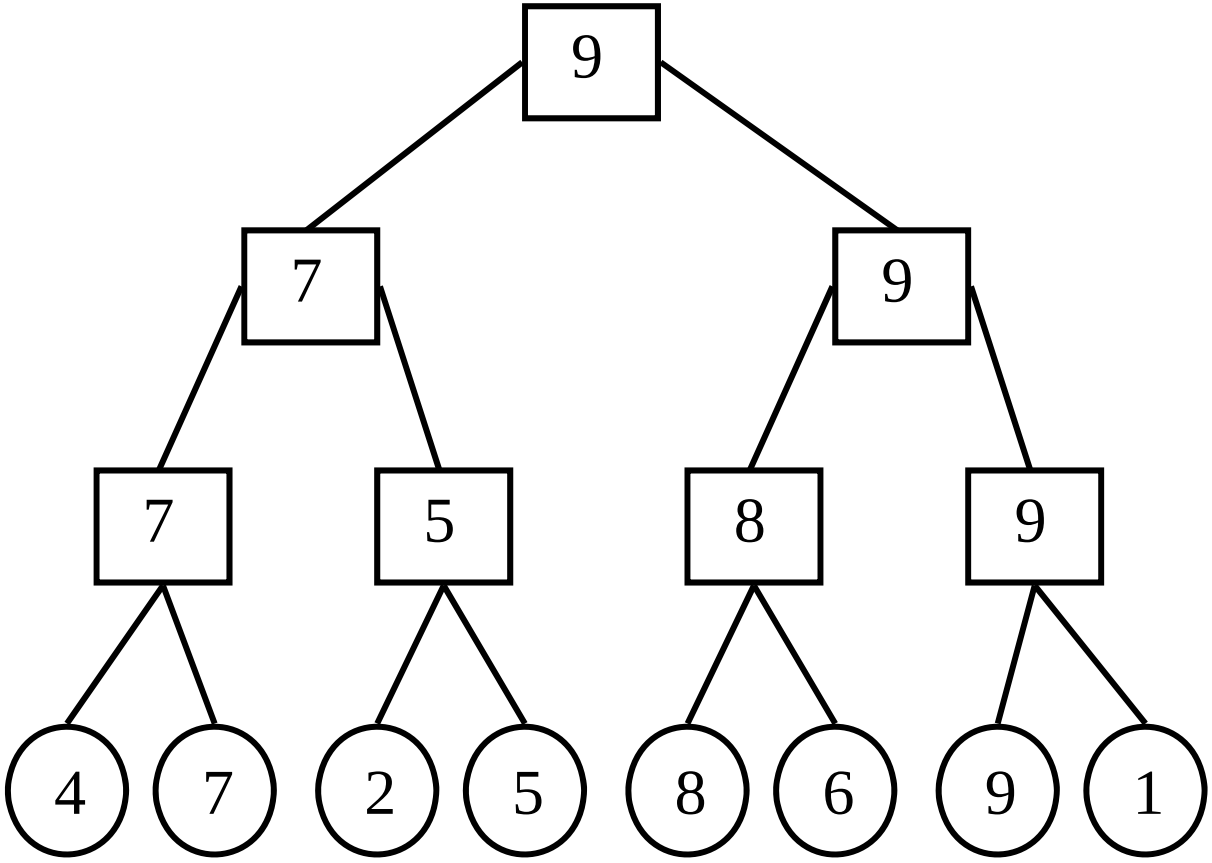
并行算法的基本设计技术

- 平衡树方法（**Balanced Trees Method**）
- 倍增技术（**Doubling Techniques**）
- 分治策略（**Divide-and-Conquer Strategy**）
- 划分原理（**Partitioning Principle**）
- 流水线技术（**Pipelining Techniques**）

平衡树方法

- 设计思想
  - 树叶结点为输入，中间结点为处理结点，由叶向根或由根向叶逐层并行处理。
- 示例
  - 求最大值
  - 计算前缀和

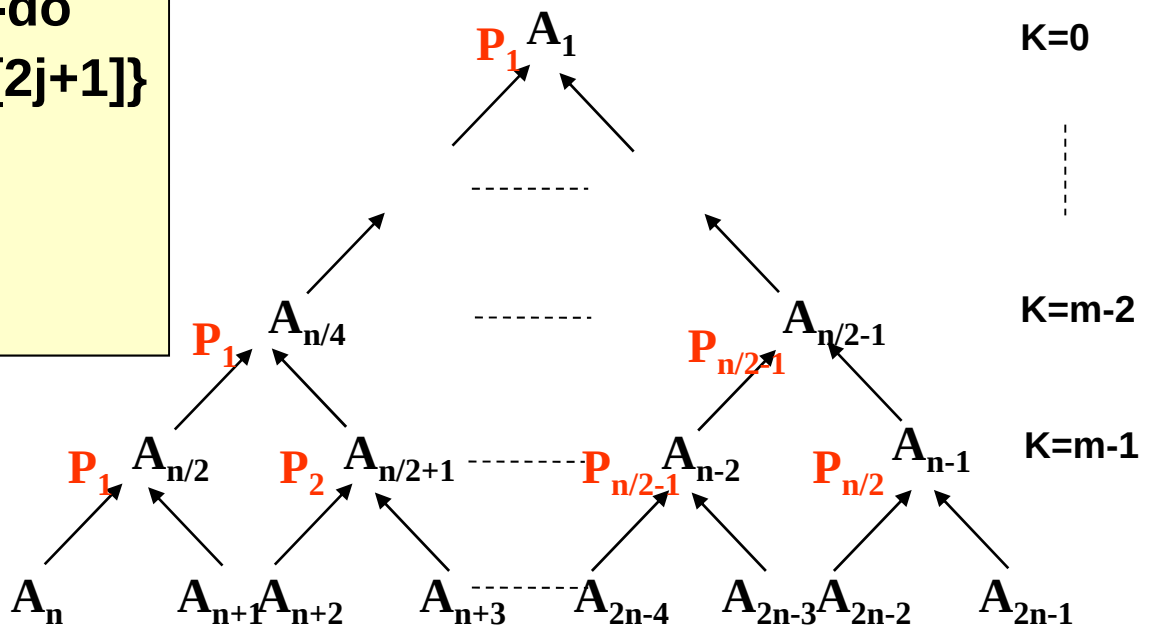
求最大值示例



求最大值

- 算法6.8：SIMD-TC(SM)上求最大值算法

```
Begin
   for k=m-1 to 0 do
      for j=2^k to 2^(k+1)-1 par-do
         A[j]=max{A[2j], A[2j+1]}
      end for
   end for
end
```



时间分析：

t(n)=m×O(1)=O(logn)　p(n)=n/2

c(n)=O(nlogn) 非成本最优

前缀和

- 问题定义

  n个元素$\{x_1,x_2,\ldots,x_n\}$，前缀和是n个部分和：

  $S_i=x_1*x_2*\ldots*x_i$，$1\leqslant i\leqslant n$ 这里*可以是＋或×

- 串行算法：$S_i=S_{i-1}*x_i$ 计算时间为 O(n)

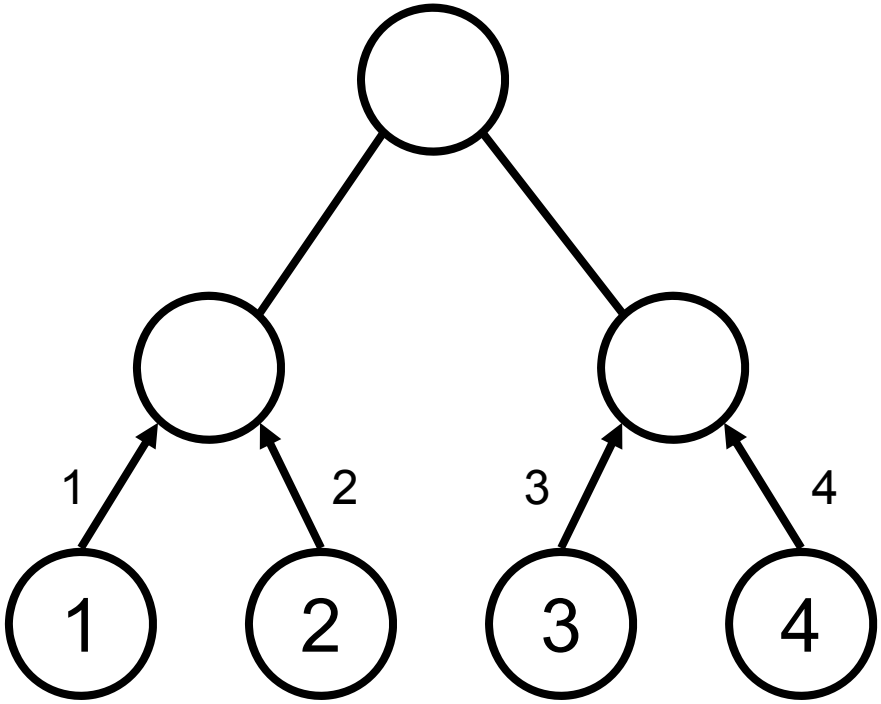
- 并行算法：算法6.9 非递归算法

  令$A[i]=x_i$, i=1~n, B[h,j]和C[h,j]为辅助数组($h=0\sim\log n$, $j=1\sim n/2^h$)

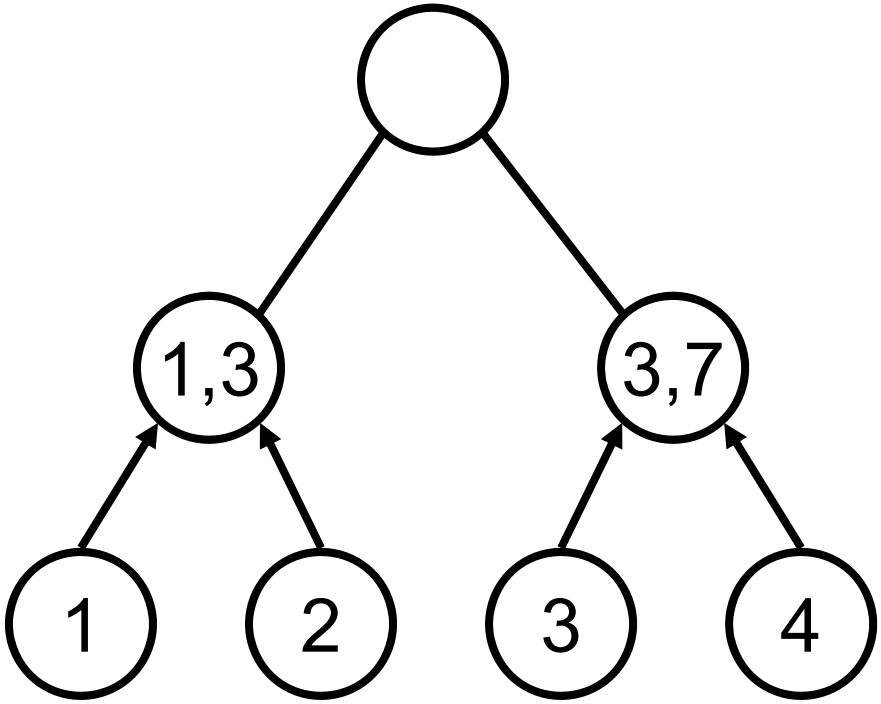
  数组B记录由叶到根正向遍历树中各结点的信息－求和

  数组C记录由根到叶反向遍历树中各结点的信息－播送前缀和

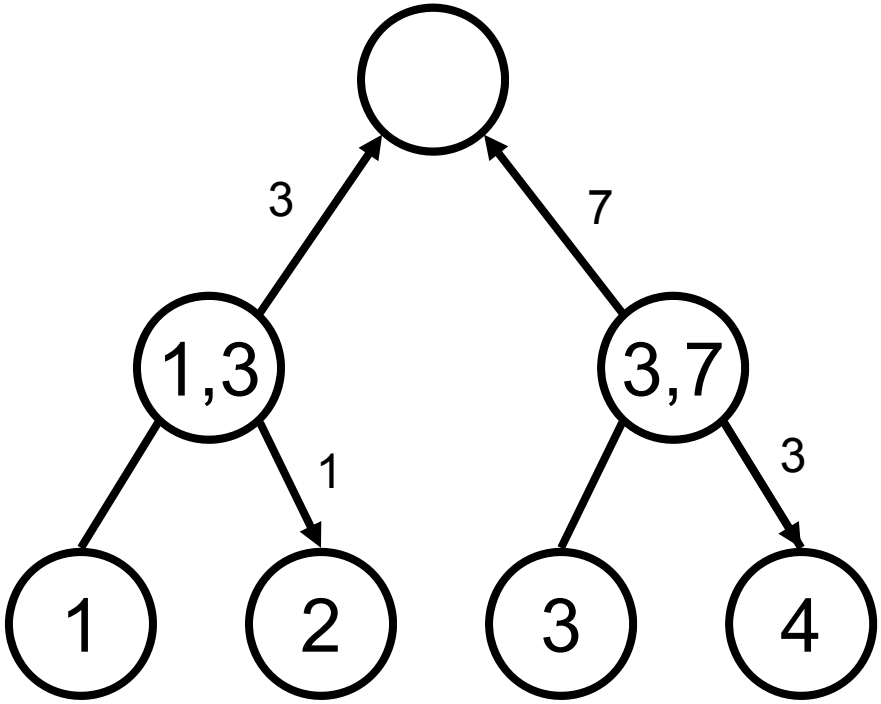
计算前缀和示例（1）



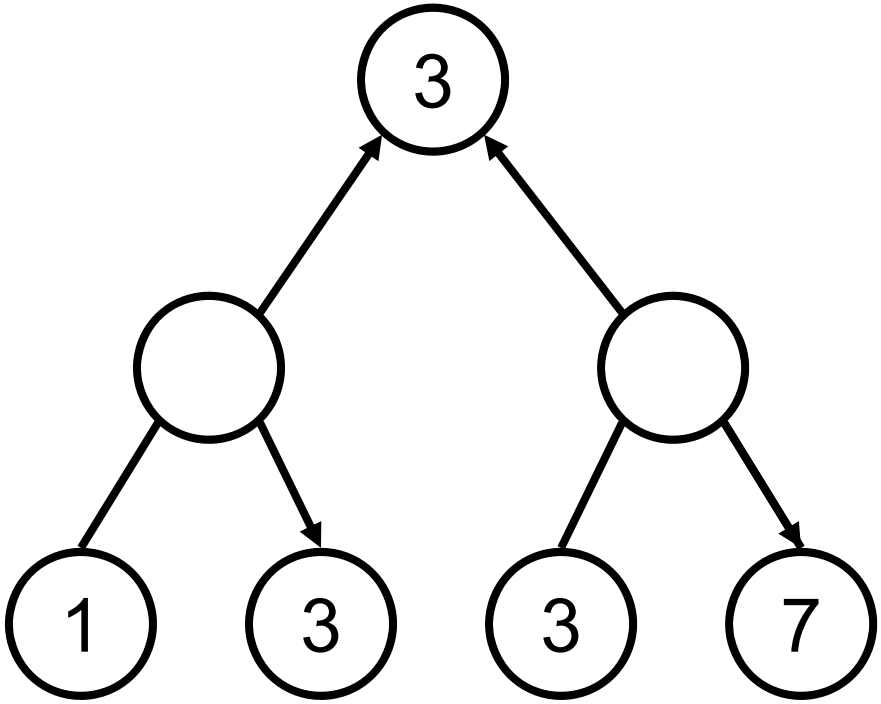
计算前缀和示例（2）



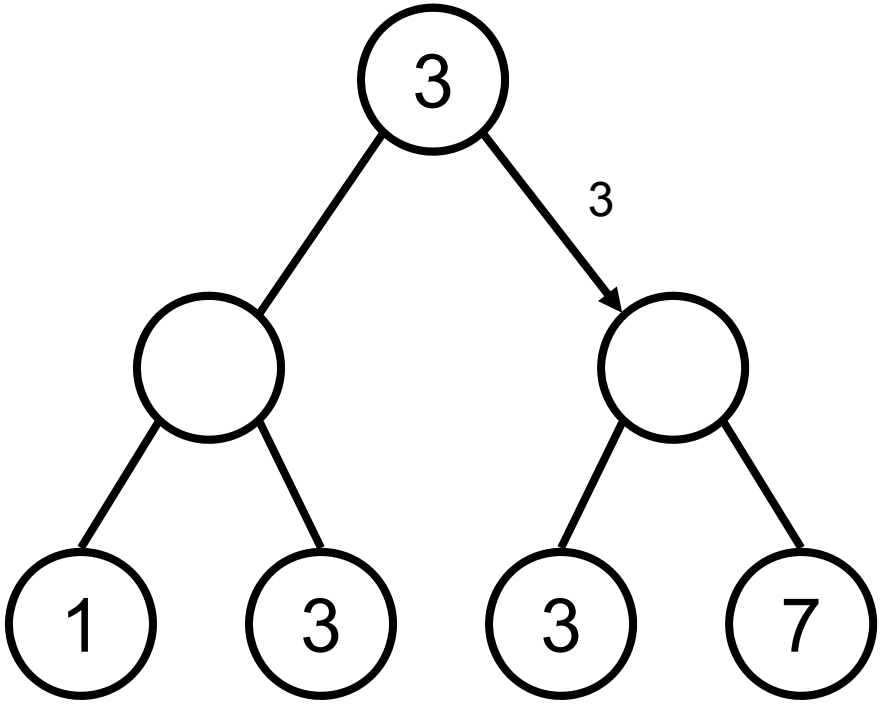
计算前缀和示例（3）



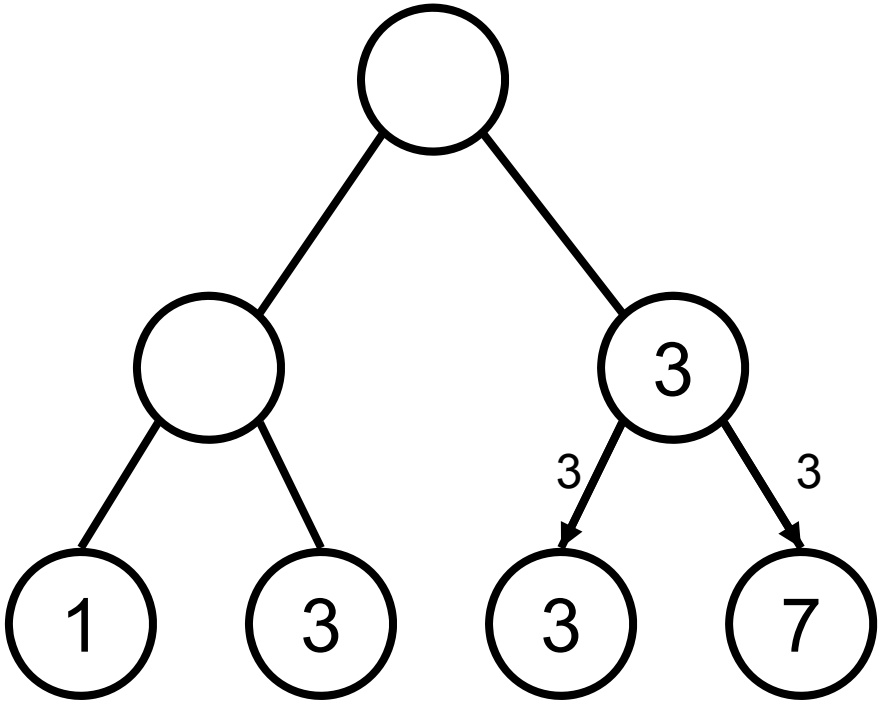
计算前缀和示例（4）



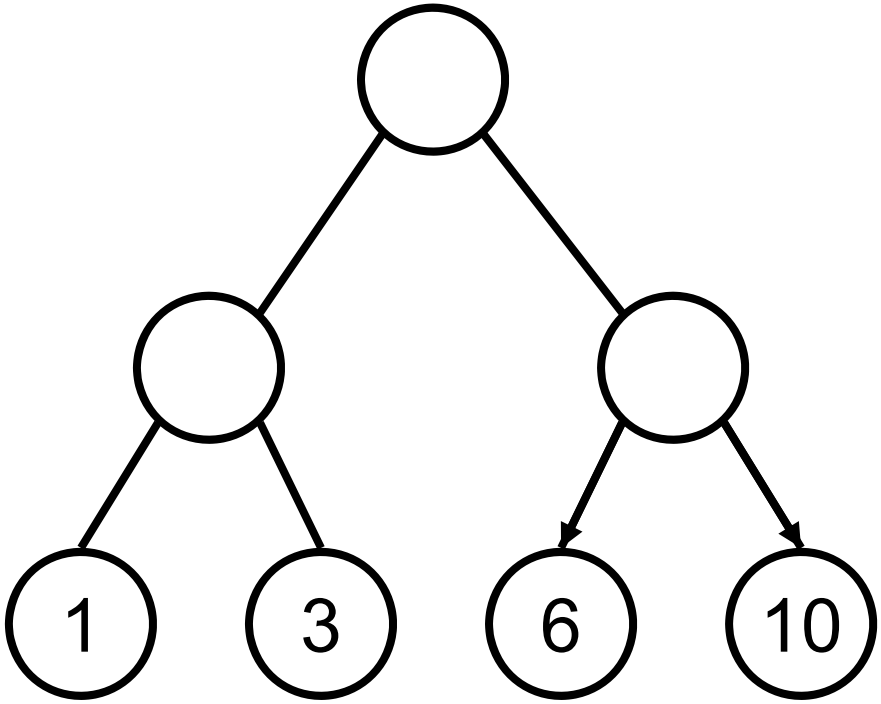
计算前缀和示例（5）



计算前缀和示例（6）

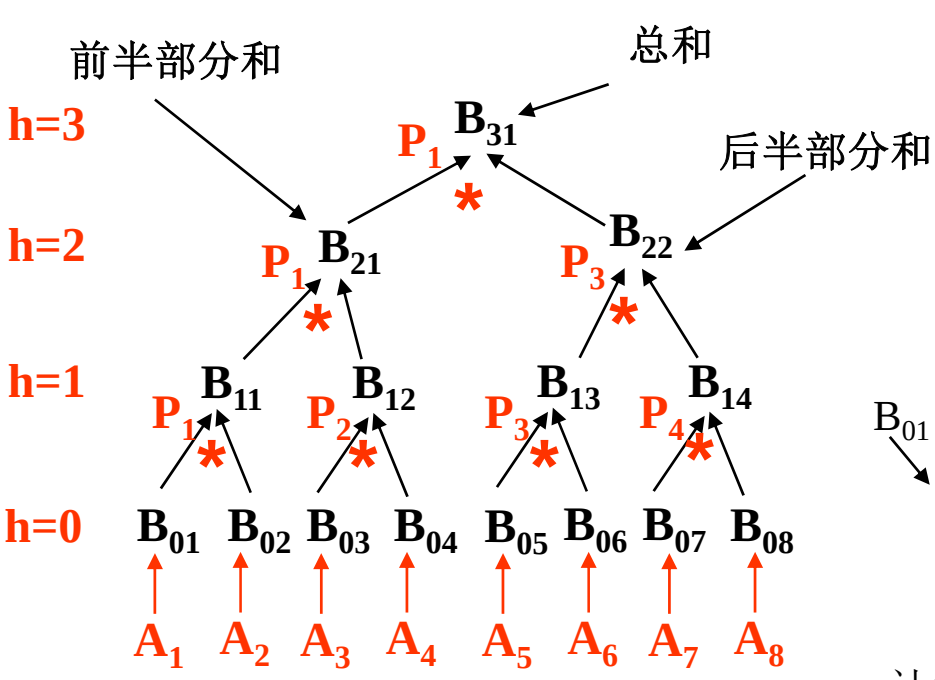


计算前缀和示例（7）

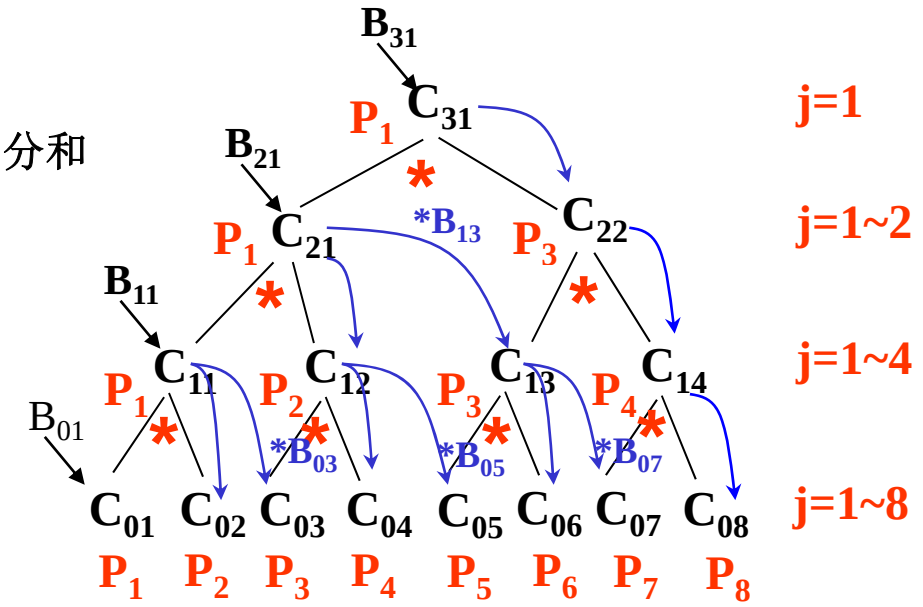


计算前缀和

- 例：n=8, p=8, C_{01}~C_{08}为前缀和



计算：B[h,j]=B[h-1,2j-1]*B[h-1,2j]
正向遍历(求和)



计算：C[h,1]=B[h,1] j=1
C[h,j]=C[h+1, j/2] j=even
C[h,j]=C[h+1,(j-1)/2]*B[h, j] j=odd>1
反向遍历(播送前缀和)

非递归求前缀和算法

算法6.9 SIMD-SM上非递归算法

```
begin
   (1 )for j=1 to n par-do    //初始化
         B[0,j]=A[j]
       end if
    (2) for h=1 to logn do   //正向遍历
         for j=1 to n/2^h par-do
            B[h,j]=B[h-1,2j-1]*B[h-1,2j]
         end for
        end for
```

```
(3) for h=logn to 0 do   //反向遍历
       for j=1 to n/2^h par-do
         (i)  if j=even then   //该节点为其父节点的右儿子
                C[h,j]=C[h+1,j/2]
              end if
         (ii) if j=1 then       //该结点为最左节点
                C[h,1]=B[h,1]
              end if
         (iii) if j=odd>1 then   //该节点为其父节点的左儿子
                C[h,j]=C[h+1,(j-1)/2]*B[h,j]
               end if
       end for
   end for
end
```

时间分析:

(1) O(1)　(2) O(logn)　(3) O(logn)

===> t(n)=O(logn) , p(n)=n , c(n)=O(nlogn)

并行算法的基本设计技术

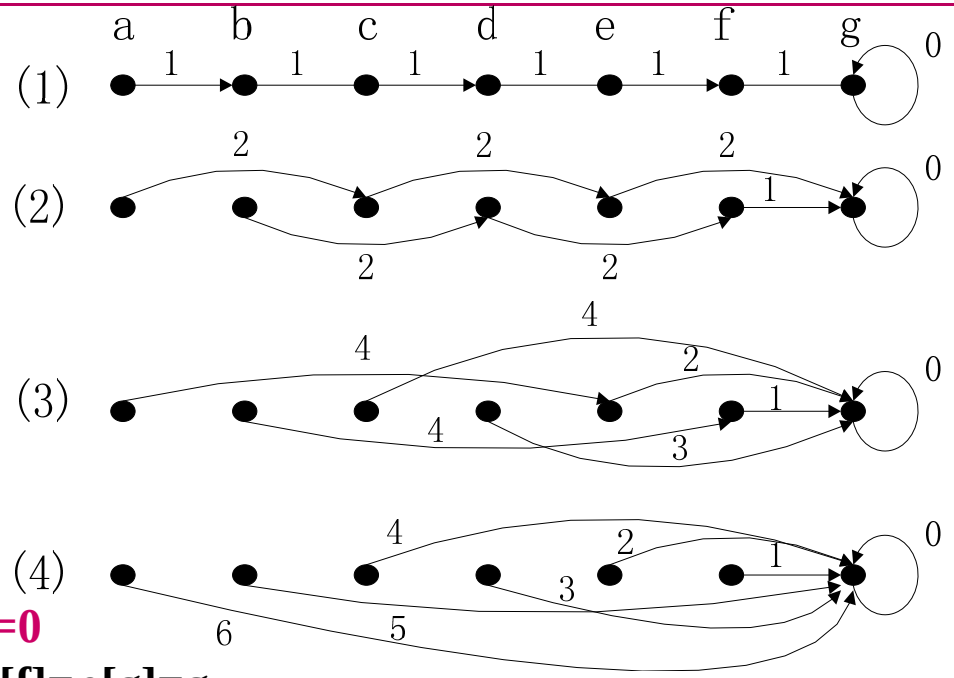
- 平衡树方法（**Balanced Trees Method**）
- 倍增技术（**Doubling Techniques**）
- 分治策略（**Divide-and-Conquer Strategy**）
- 划分原理（**Partitioning Principle**）
- 流水线技术（**Pipelining Techniques**）

倍增技术

- 设计思想
  - 又称指针跳跃(**pointer jumping**)技术，特别适合于处理链表或有向树之类的数据结构；
  - 当递归调用时，所要处理数据之间的距离逐步加倍，经过**k**步后即可完成距离为2^k的所有数据的计算。
- 示例
  - 表序问题
  - 求森林的根

表序问题

- 问题描述
  - n个元素的列表L，求出每个元素在L中的次第号（秩或位序或rank(k)），rank(k)可视为元素k至表尾的距离；



- 示例： n=7

(1) p[a]=b, p[b]=c, p[c]=d, p[d]=e,
p[e]=f, p[f]=g, p[g]=g
r[a]=r[b]=r[c]=r[d]=r[e]=r[f]=1, r[g]=0

(2) p[a]=c, p[b]=d, p[c]=e, p[d]=f, p[e]=p[f]=p[g]=g
r[a]=r[b]=r[c]=r[d]=r[e]=2, r[f]=1, r[g]=0

(3) p[a]=e, p[b]=f, p[c]=p[d]=p[e]=p[f]=p[g]=g
r[a]=4, r[b]=4, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0

注：递归计算位序r

(4) p[a]=p[b]=p[c]=p[d]=p[e]=p[f]=p[g]=g
r[a]=6, r[b]=5, r[c]=4, r[d]=3, r[e]=2, r[f]=1, r[g]=0

求元素表序算法

(1) 并行做：初始化p[k]和distance[k] //O(1)

(2) 执行⌈logn⌉次 //O(logn)

(2.1) 对k并行地做 //O(1)

如果k的后继不等于k的后继之后继，则

(i) distance[k]= distance[k]+ distance[p[k]]

(ii) p[k]=p[p[k]]

(2.2) 对k并行地做

rank[k]=distance[k] //O(1)

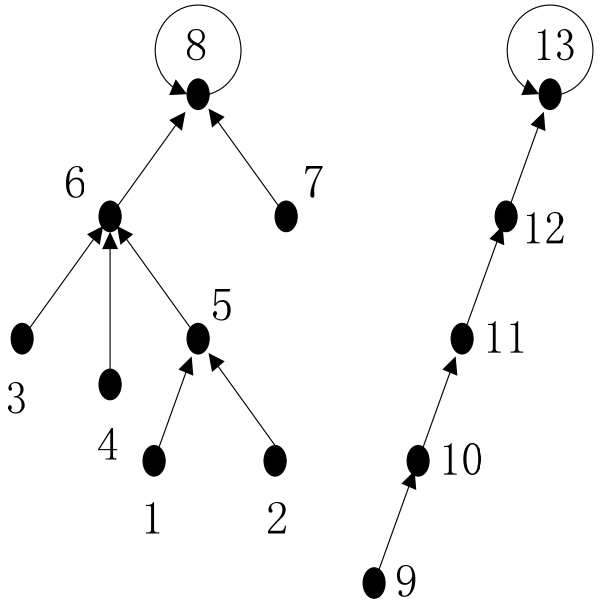
运行时间：t(n)=O(logn) p(n)=n （算法6.10）

求森林的根（1）

- 问题描述

一组有向树F中, 如果<i, j>是F中的一条弧，则p[i]=j（即j是i的双亲）；若i为根，则p[i]=i。求每个结点j(j=1~n)的树根s[j].

- 示例



(a)

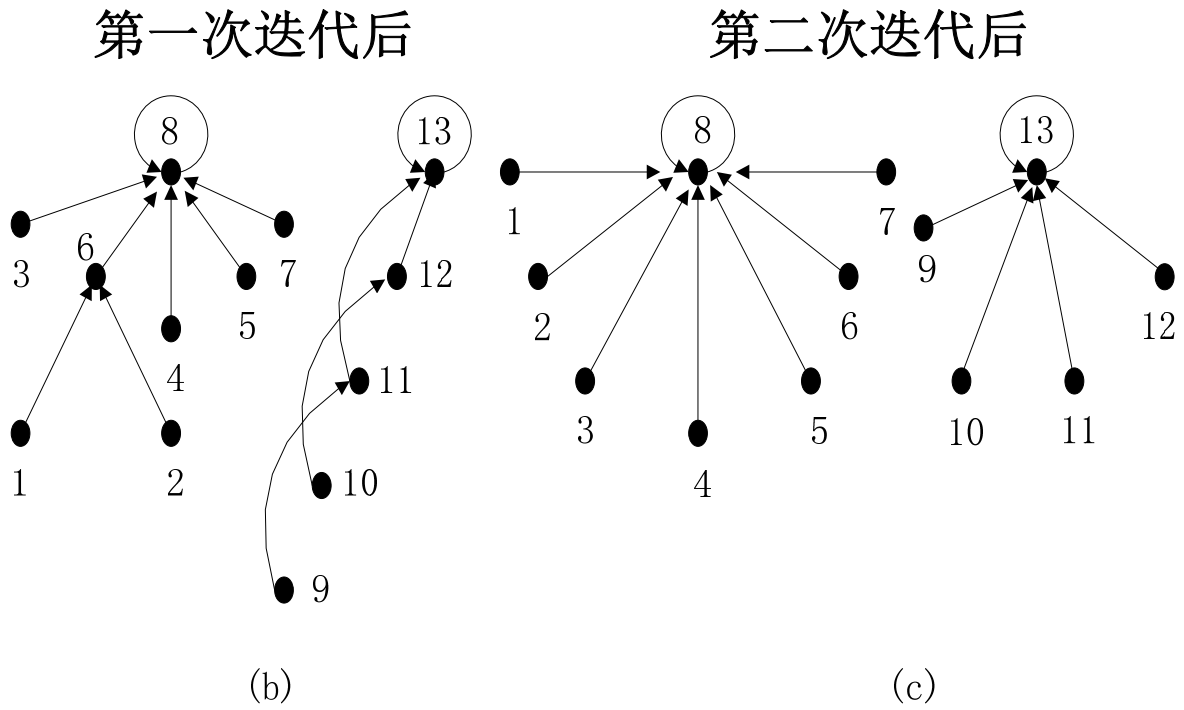
初始时：

P[1]=p[2]=5　p[3]=p[4]=p[5]=6
P[6]=p[7]=8　p[8]=8

P[9]=10　p[10]=11 p[11]=12
p[12]=13　p[13]=13

求森林的根（2）

- 迭代



运行时间：**t(n)=O(logn)** （算法**6.11**）

并行算法的基本设计技术

- 平衡树方法（**Balanced Trees Method**）
- 倍增技术（**Doubling Techniques**）
- 分治策略（**Divide-and-Conquer Strategy**）
- 划分原理（**Partitioning Principle**）
- 流水线技术（**Pipelining Techniques**）

分治策略

- 设计思想
  - 将原问题划分成若干个相同的子问题分而治之，若子问题仍然较大，则可以反复递归应用分治策略处理这些子问题，直至子问题易求解。
- 求解步骤
  - 将输入划分成若干个规模相等的子问题；
  - 同时(并行地)递归求解这些子问题；
  - 并行地归并子问题的解成为原问题的解。
- 示例
  - **SIMD-SM模型上的FFT递归算法**

FFT递归算法

- **DFT离散富里叶变换的定义**

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 $\omega = e^{2\pi i/n}$，$i = \sqrt{-1}$

写成矩阵形式为

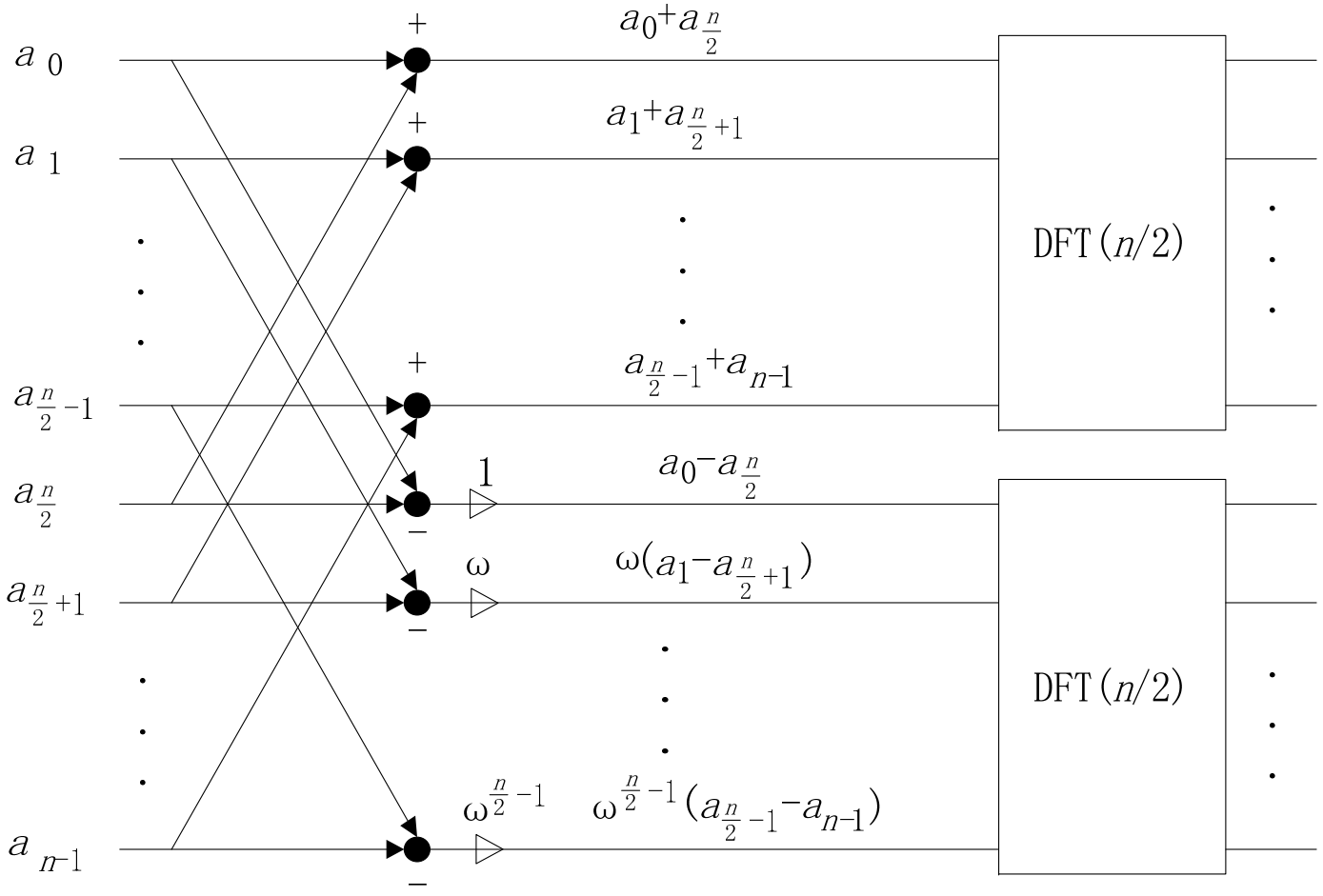
$$\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)}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

时间分析：串行直接计算DFT需要$O(n^2)$

FFT递归算法

FFT递归算法： 将原问题的DFT划分为两个规模为n/2的子问题的DFT

- **SIMD-SM模型上的算法：图11.3**



并行算法的基本设计技术

- 平衡树方法（**Balanced Trees Method**）
- 倍增技术（**Doubling Techniques**）
- 分治策略（**Divide-and-Conquer Strategy**）
- 划分原理（**Partitioning Principle**）
- 流水线技术（**Pipelining Techniques**）

划分原理

- 设计思想
  - 将原问题划分成p个独立的规模几乎相等的子问题
  - p台处理器并行地求解各子问题
- 划分重点在于：子问题易解，组合成原问题的解方便
- 常见划分方法
  - 均匀划分
  - 方根划分
  - 对数划分
  - 功能划分

均匀划分

- 方法: n个元素A[1..n]分成p组，每组A[(i-1)n/p+1..in/p]，i=1~p
- 示例: MIMD-SM模型上的PSRS排序

begin

（1）均匀划分：将n个元素A[1..n]均匀划分成p段，每个p_i处理A[(i-1)n/p+1..in/p]

（2）局部排序：p_i调用串行排序算法对A[(i-1)n/p+1..in/p]排序

（3）选取样本：p_i从其有序子序列A[(i-1)n/p+1..in/p]中选取p个样本元素

（4）样本排序：用一台处理器对p^2个样本元素进行串行排序

（5）选择主元：用一台处理器从排好序的样本序列中选取p-1个主元，并播送给其他p_i

（6）主元划分：p_i按主元将有序段A[(i-1)n/p+1..in/p]划分成p段

（7）全局交换：各处理器将其有序段按段号交换到对应的处理器中

（8）归并排序：各处理器对接收到的元素进行归并排序

end.

PSRS排序算法

- 例 PSRS排序过程。N=27， p=3，PSRS排序如下：

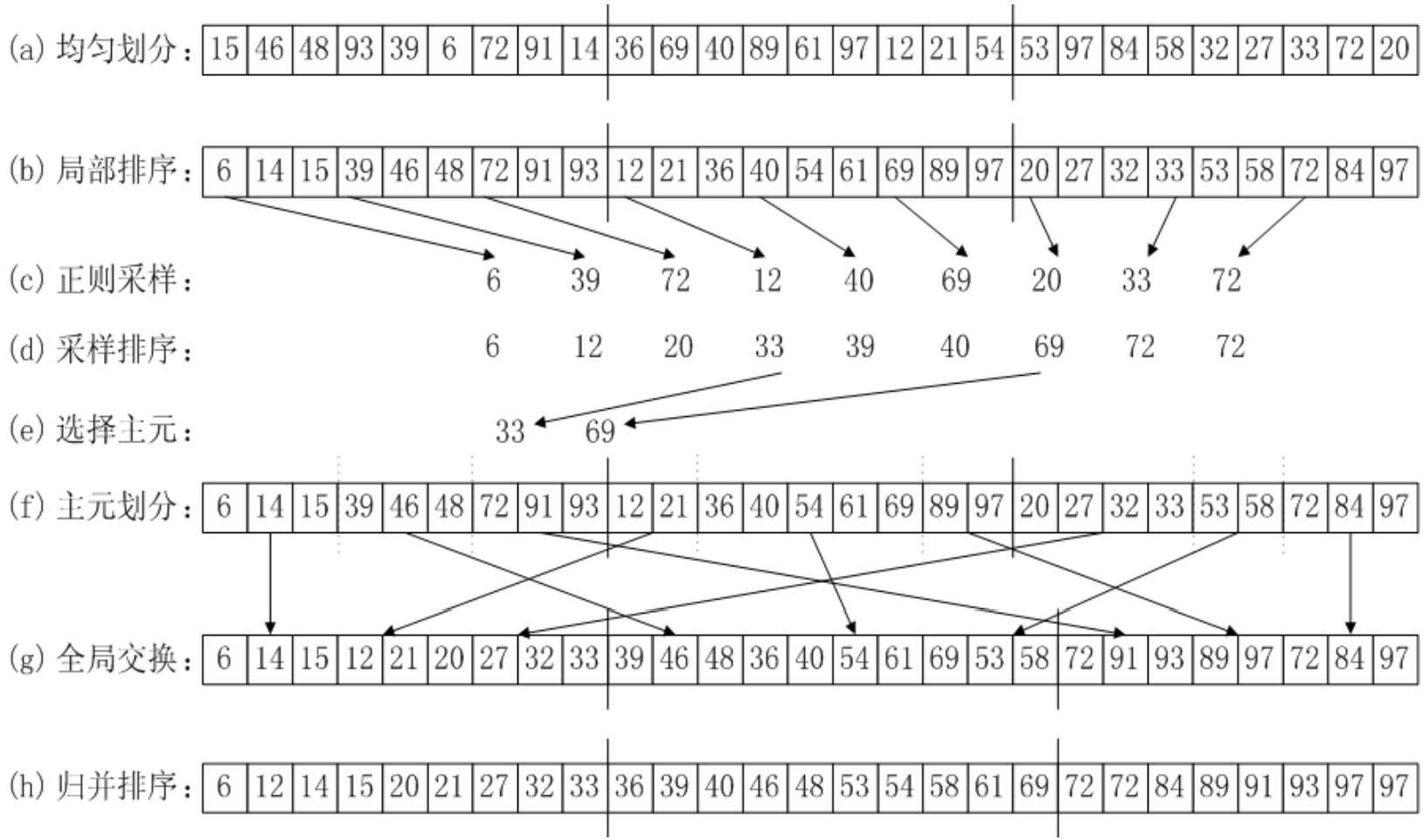


图6.1

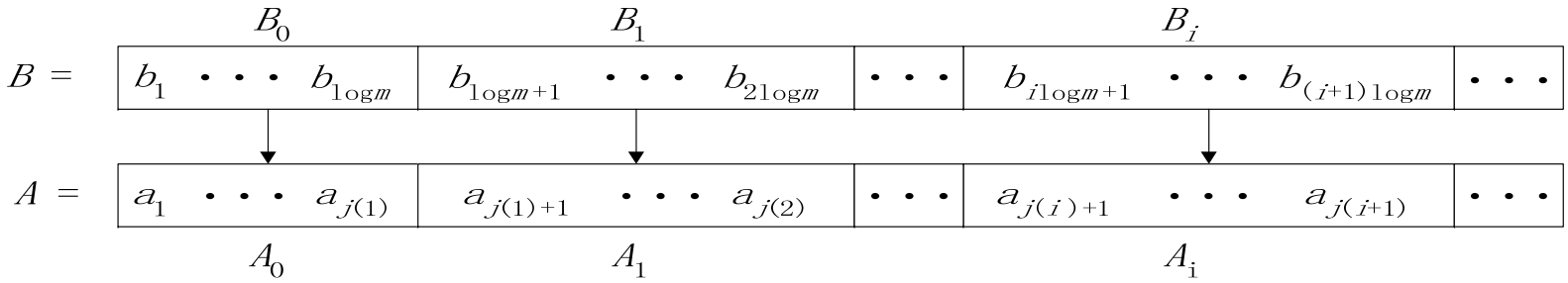
对数划分

- 划分方法

  n个元素A[1..n]分成A[(i-1)logn+1..ilogn]，i=1~n/logn

- 示例：**PRAM-CREW上的对数划分并行归并排序**

（1）归并过程: 设有序组A[1..n]和B[1..m]



j[i]=rank(b_{ilogm}:A)为b_{ilogm}在A中的位序，即A中小于等于b_{ilogm}的元素个数

（2）例：A=(4,6,7,10,12,15,18,20), B=(3,9,16,21) n=8, m=4

=>logm=log4=2

=> j[1]=rank(b_{logm}:A)=rank(b_2:A)=rank(9:A)=3, j[2]=…=8

B_0: 3, 9 B_1: 16, 21

A_0: 4, 6, 7 A_1: 10, 12, 15, 18, 20

A和B归并化为(A_0, B_0)和(A_1, B_1)的归并

方根划分

- 方法: *n*个元素A[1..*n*]分成A[(i-1)n^(1/2)+1..in^(1/2)]，i=1~n^(1/2), p=n^(1/2)
- 示例: SIMD-CREW模型上的Valiant归并排序算法(1975年发表)

//有序组A[1..p]、B[1..q], (假设p<=q), 处理器数k=(pq)^(1/2)

begin

(1)方根划分：A,B分别按*ip*^(1/2)和*iq*^(1/2)分成若干段；

(2)段间比较：A划分元与B划分元比较(共有p^(1/2)q^(1/2)对)，确定A划分元应插入B中的区段；

(3)段内比较：A划分元与B相应段内元素进行比较，并插入适当的位置；

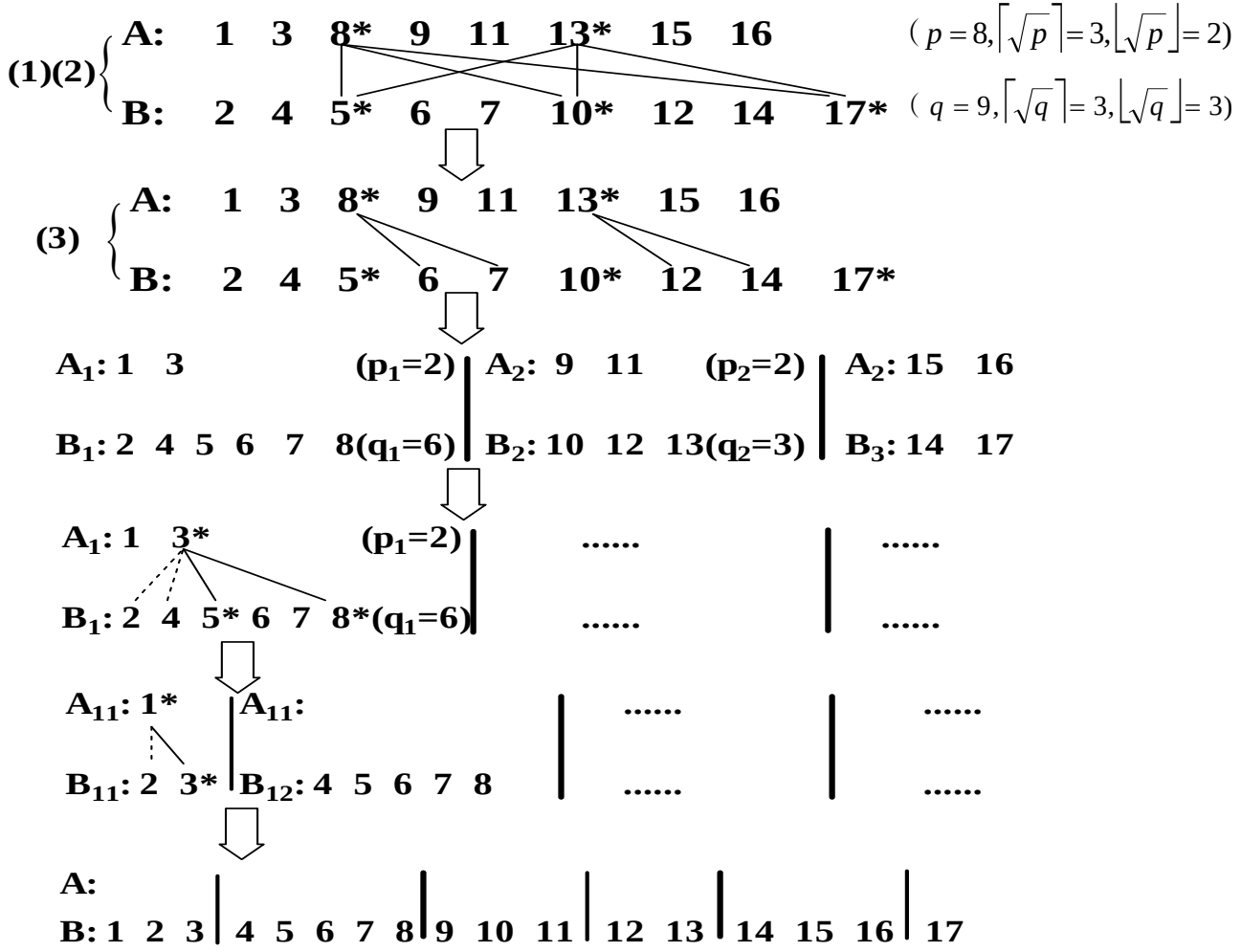
(4)递归归并：B按插入的A划分元重新分段，与A相应段(A除去原划分元)构成了成对的段组，对每对段组递归执行(1)~(3)，直至A组为0时，递归结束

end.

时间复杂度：若p=q=n，t(n)=O(loglogn)　p(n)=n

方根划分技术

- 示例: A={1,3,8,9,11,13,15,16},p=8; B={2,4,5,6,7,10,12,14,17},q=9



功能划分技术

- 划分方法

  n个元素A[1..n]分成等长的p组，每组满足某种特性。

- 示例：(m, n)选择问题(求出n个元素中前m个最小者)
  - 功能划分：要求每组元素个数必须大于m；
  - 算法：算法6.4

    输入：$A=(a_1,\ldots,a_n)$；输出：前m个最小者；

    Begin

    (1) 功能划分：将A划分成g=n/m组，每组含m个元素；

    (2) 局部排序：将各组并行进行排序；

    (3) 两两比较：将所排序的各组两两进行比较，从而形成MIN序列；

    (4) 排序-比较：对各个MIN序列，重复执行第(2)和第(3)步，直至选出m个最小者。

    End

功能划分技术

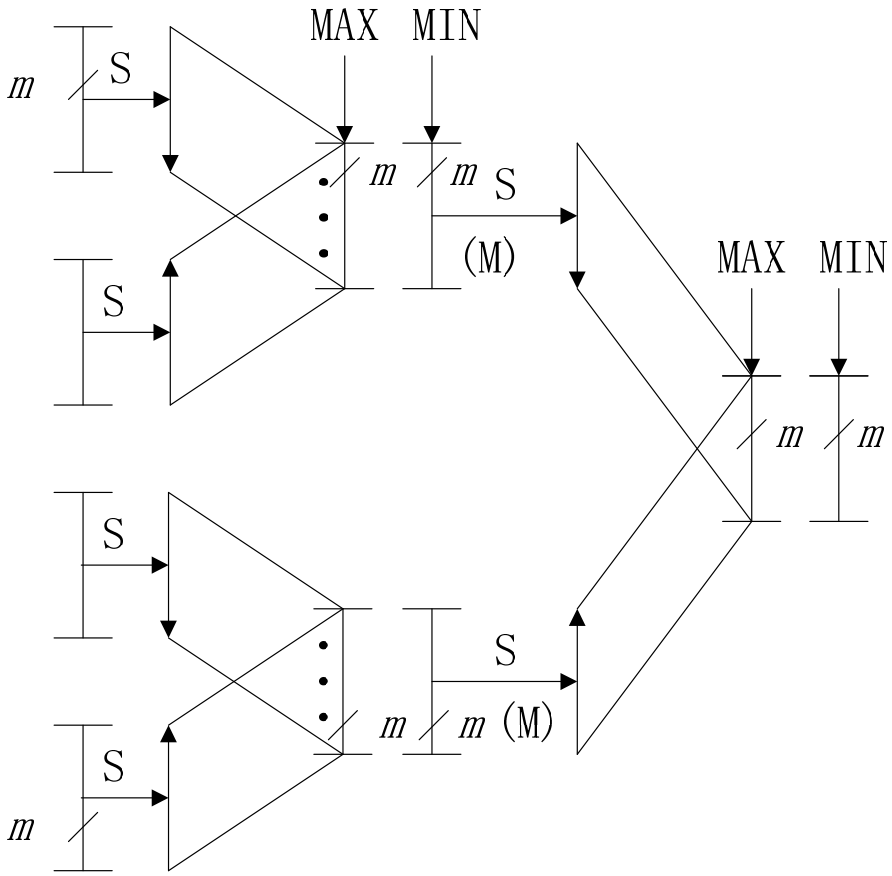


图6.3 (m-n)-选择过程

并行算法的基本设计技术

- 平衡树方法（**Balanced Trees Method**）
- 倍增技术（**Doubling Techniques**）
- 分治策略（**Divide-and-Conquer Strategy**）
- 划分原理（**Partitioning Principle**）
- 流水线技术（**Pipelining Techniques**）

流水线技术

- 设计思想
  - 将算法流程划分成**p**个前后衔接的任务片断，每个任务片断的输出作为下一个任务片断的输入；
  - 所有任务片断按同样的速率产生出结果。
- **Remark**
  - 流水线技术是一种广泛应用在并行处理中的技术；
  - 脉动算法**(Systolic algorithm)**是其中一种流水线技术；

5-point DFT的计算

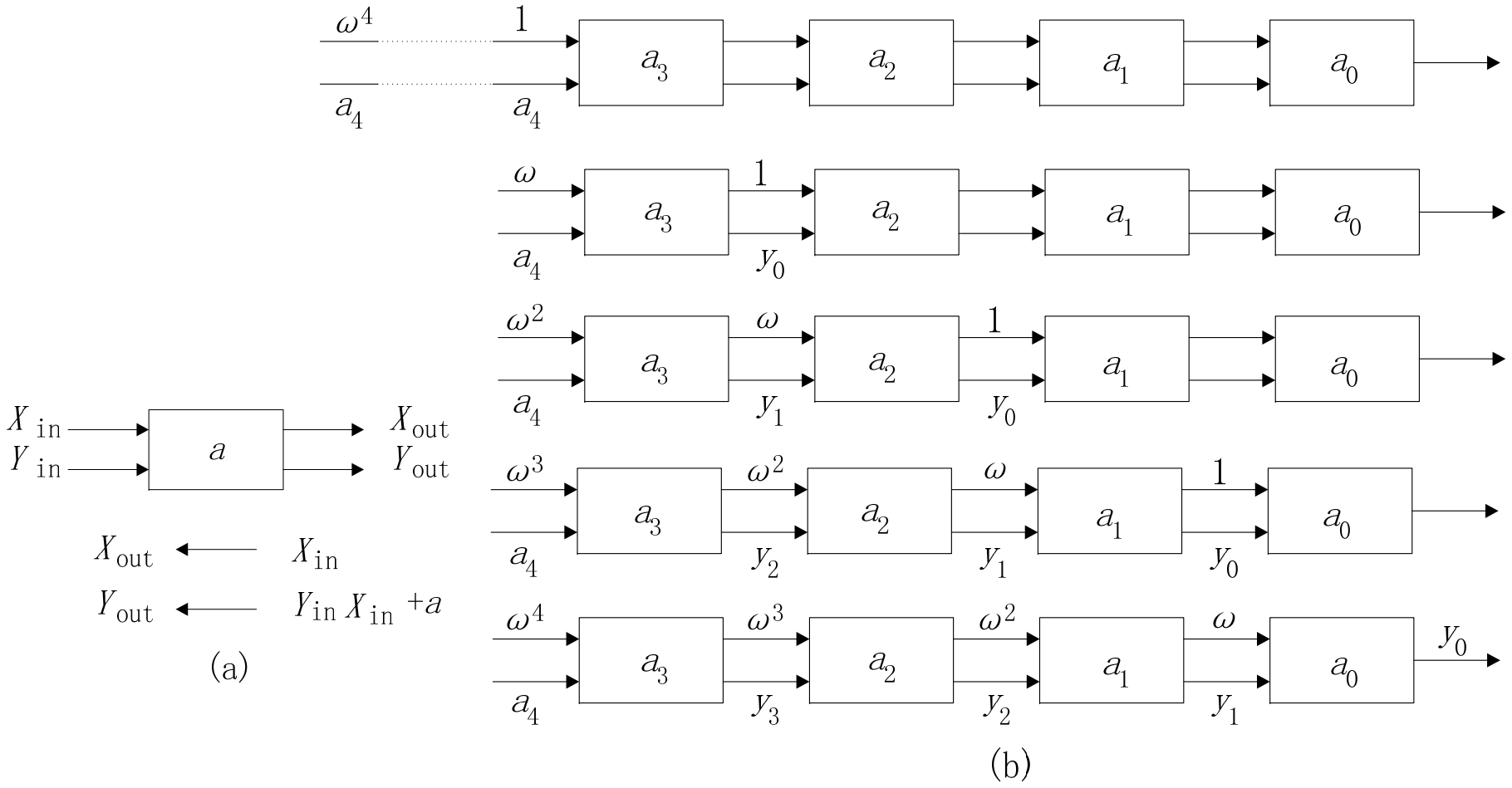
- 问题描述

  5-point DFT（离散傅里叶变换）的计算。应用秦九韶**(Horner)**法则，

$$\begin{cases} y_0 = b_0 = a_4\omega^0 + a_3\omega^0 + a_2\omega^0 + a_1\omega^0 + a_0 \\ y_1 = b_1 = a_4\omega^4 + a_3\omega^3 + a_2\omega^2 + a_1\omega^1 + a_0 \\ y_2 = b_2 = a_4\omega^8 + a_3\omega^6 + a_2\omega^4 + a_1\omega^2 + a_0 \\ y_3 = b_3 = a_4\omega^{12} + a_3\omega^9 + a_2\omega^6 + a_1\omega^3 + a_0 \\ y_4 = b_4 = a_4\omega^{16} + a_3\omega^{12} + a_2\omega^8 + a_1\omega^4 + a_0 \end{cases} \Rightarrow \begin{cases} y_0 = (((a_4\omega^0 + a_3)\omega^0 + a_2)\omega^0 + a_1)\omega^0 + a_0 \\ y_1 = (((a_4\omega^1 + a_3)\omega^1 + a_2)\omega^1 + a_1)\omega^1 + a_0 \\ y_2 = (((a_4\omega^2 + a_3)\omega^2 + a_2)\omega^2 + a_1)\omega^2 + a_0 \\ y_3 = (((a_4\omega^3 + a_3)\omega^3 + a_2)\omega^3 + a_1)\omega^3 + a_0 \\ y_4 = (((a_4\omega^4 + a_3)\omega^4 + a_2)\omega^4 + a_1)\omega^4 + a_0 \end{cases}$$

DFT计算

- 示例： **5-point DFT的计算，p(n)=n-1, t(n)=2n-2=O(n)**



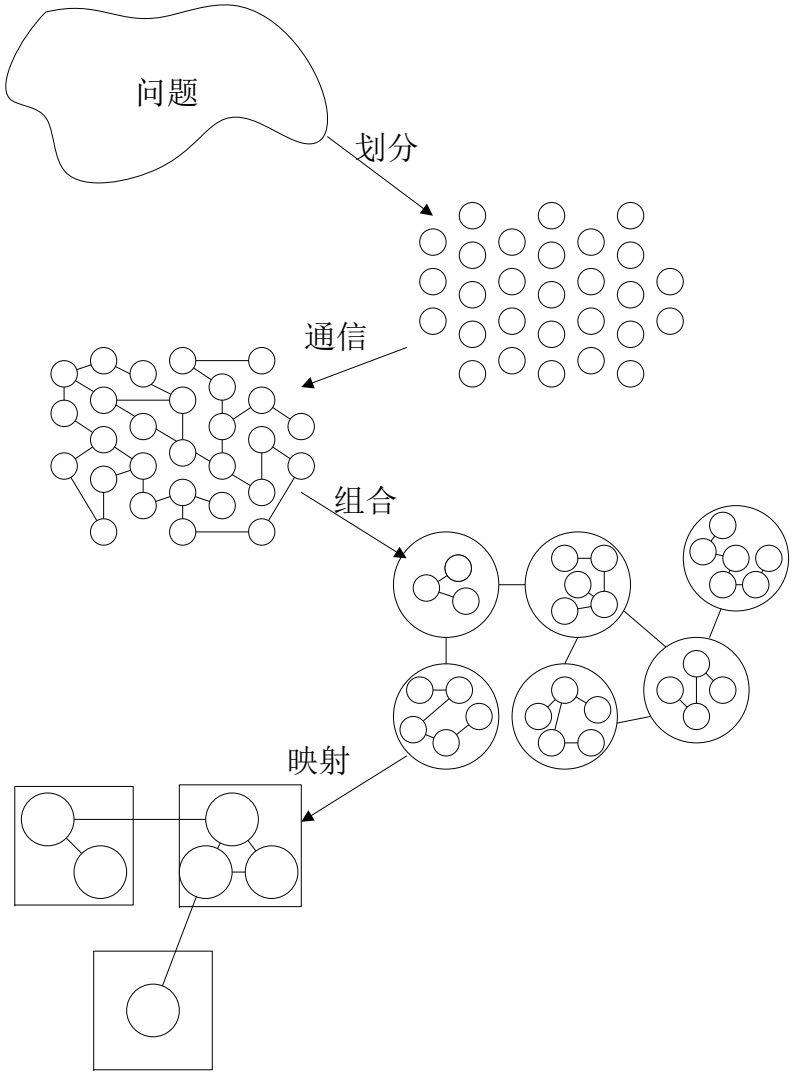
内容概要

- 并行算法的基本设计技术
- 并行算法的一般设计过程
  - 划分
  - 通信
  - 组合
  - 映射

PCAM设计方法学

- 设计并行算法的四个阶段
  - 划分（**Partitioning**）
  - 通信（**Communication**）
  - 组合（**Agglomeration**）
  - 映射（**Mapping**）
- 划分：分解成小的任务，开拓并发性；
- 通信：确定诸任务间的数据交换，监测划分的合理性；
- 组合：依据任务的局部性，组合成更大的任务；
- 映射：将每个任务分配到处理器上，提高算法的性能。

PCAM设计过程



内容概要

- 并行算法的一般设计过程
  - 划分
  - 通信
  - 组合
  - 映射

划分方法描述

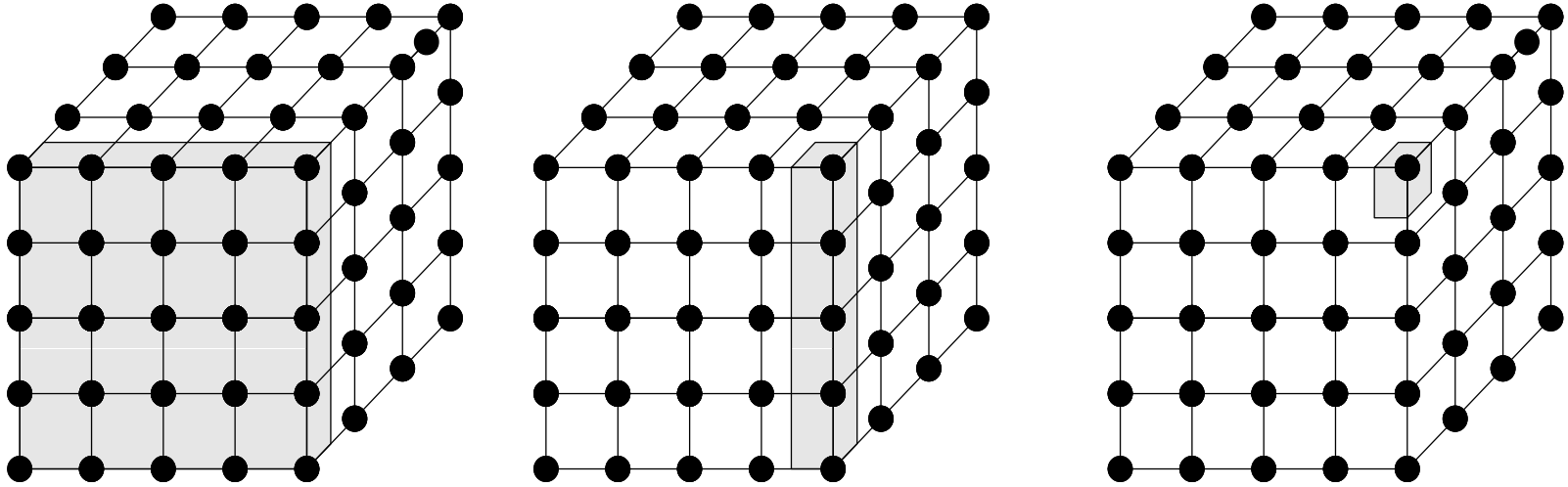
- 充分开拓算法的并发性和可扩展性
- 分为两类划分：
  - 域分解（**domain decomposition**）/数据分解
  - 功能分解（**functional decomposition**）
- 先进行数据分解（称域分解），再进行计算功能的分解（称功能分解）
- 使数据集和计算集互不相交
- 划分阶段忽略处理器数目和目标机器的体系结构

域分解

- 划分的对象是数据，可以是算法的输入数据、中间处理数据和输出数据
- 将数据分解成大致相等的小数据片
- 划分时考虑数据上的相应操作
- 如果一个任务需要别的任务中的数据，则会产生任务间的通信

域分解

- 示例：三维网格的域分解，各格点上计算都是重复的。下图是三种分解方法：

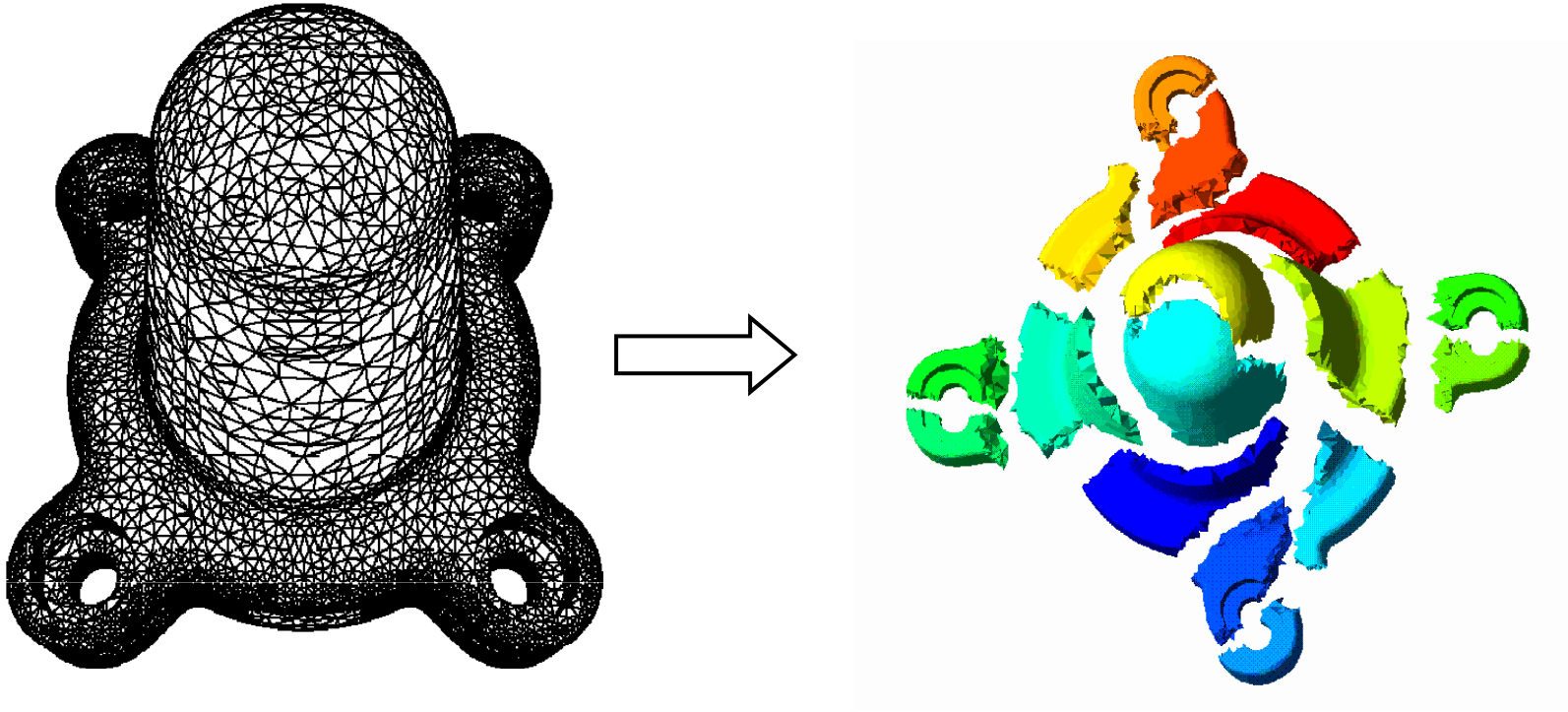


1-D　　2-D　　3-D

图7.2

域分解

- 不规则区域的分解示例：

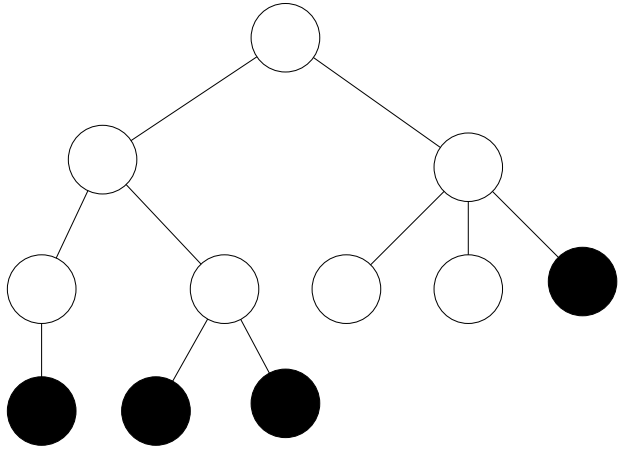


功能分解

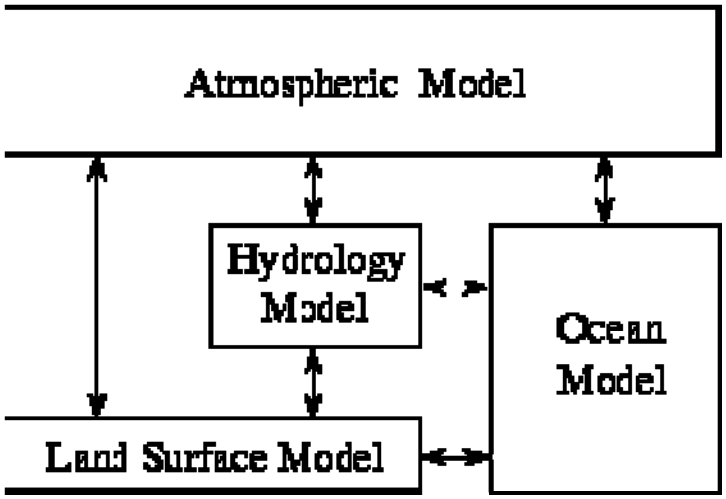
- 划分的对象是计算，将计算划分为不同的任务，其出发点不同于域分解
- 划分后，研究不同任务所需的数据。如果这些数据不相交的，则划分是成功的；如果数据有相当的重叠， 意味着要重新进行域分解和功能分解
- 功能分解是一种更深层次的分解

功能分解

- 示例1：搜索树



- 示例2：气候模型



划分判据

- 划分是否具有灵活性？
- 划分是否避免了冗余计算和存储？
- 划分任务尺寸是否大致相当？
- 任务数与问题尺寸是否成比例？
- 功能分解是一种更深层次的分解，是否合理？

内容概要

- 并行算法的基本设计技术
- 并行算法的一般设计过程
  - 划分
  - 通信
  - 组合
  - 映射

通信方法描述

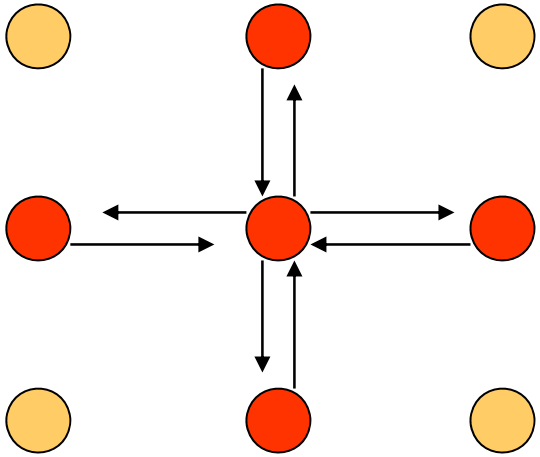
- 通信是**PCAM**设计过程的重要阶段；
- 划分产生的诸任务，一般不能完全独立执行，需要在任务间进行数据交流；从而产生了通信；
- 功能分解确定了诸任务之间的数据流；
- 诸任务是并发执行的，通信则限制了这种并发性；

四种通信模式

- 局部/全局通信（**Local/Global communication**）
- 结构化/非结构化通信（ **Structure/Unstructured communication**）
- 静态/动态通信（**Static/Dynamic communication**）
- 同步/异步通信（**Synchronous/Asynchronous communication**）

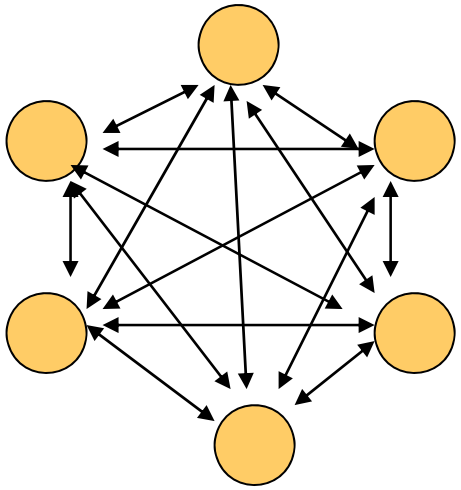
局部通信

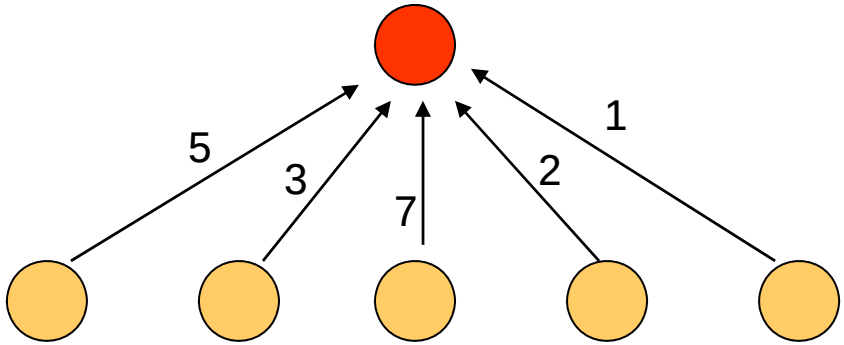
- 通信限制在一个邻域内



全局通信

- 许多任务参与，非局部的
- 例如
  - **All to All**
  - **Master-Worker**





全局通信的例子

- 例如考虑实现一个归约操作（顺序求和），根进程**S**，负责从**n**个分布式任务中一次接收一个值并相加
- 有**n=8** 个任务，**8**个通道连接到 **S** （图中数字标识表示步数）
- 这种方法需要 **O(n)**步，不优化！

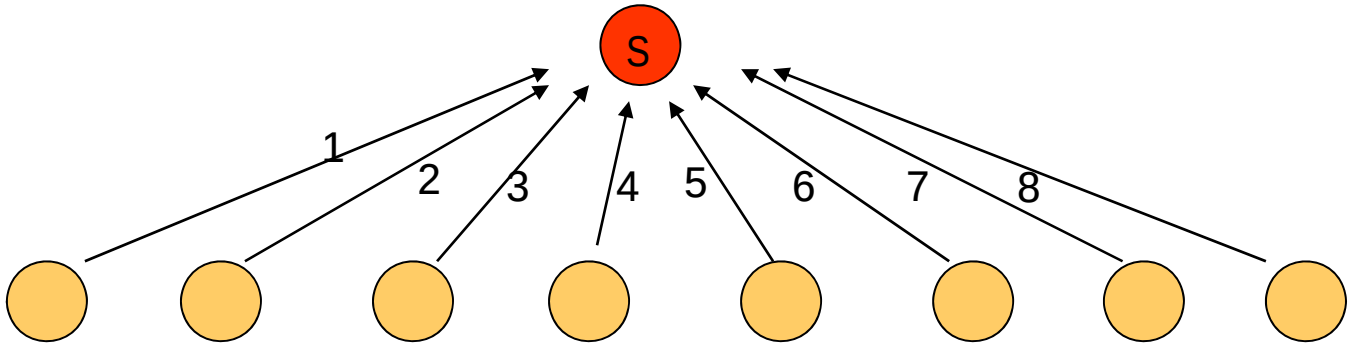
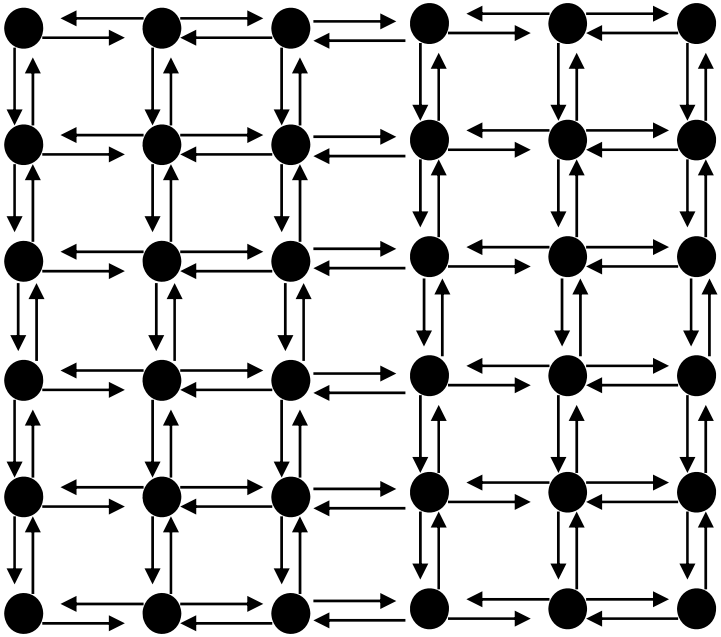
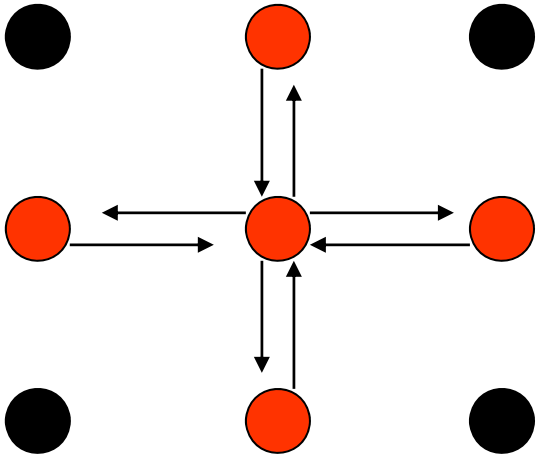


图**7.5**

结构化通信

- 每个任务的通信模式是相同的
- 下面是否存在一个相同通信模式?



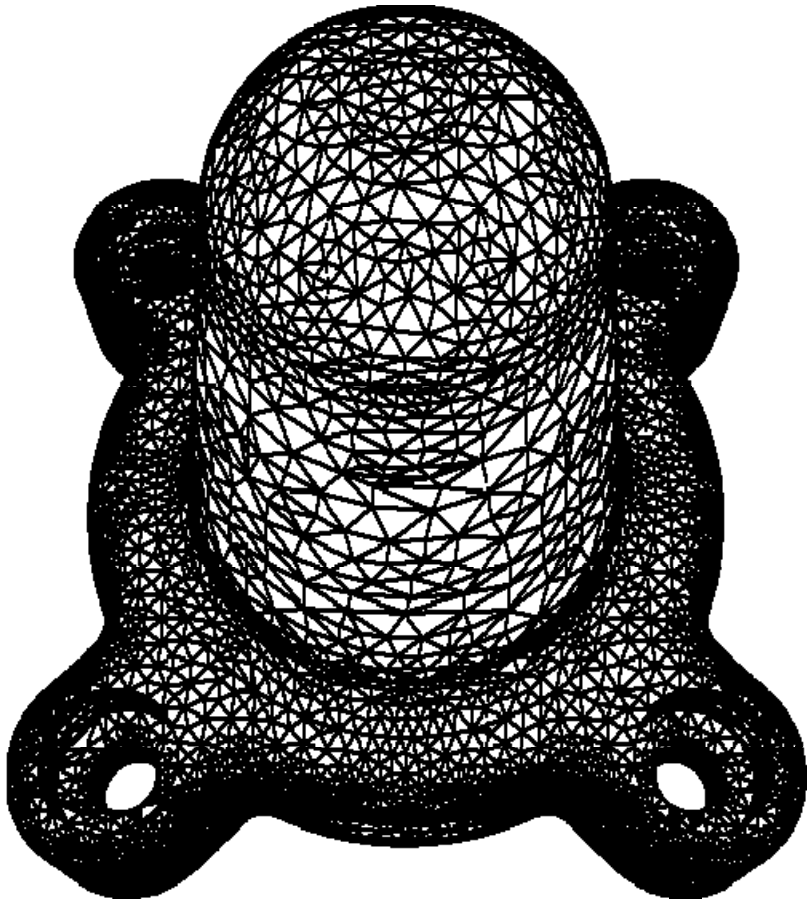


非结构化、动态和异步通信

- 没有一个统一的通信模式
- 例如：无结构化网格

- 动态通信：通信模式不规整

- 同步通信：通信双方同时产生通信操作
- 异步通信：非同步通信操作，一般由接收者请求
  - 如消息传递（**message-Passing**）



通信判据

- 所有任务是否执行大致相当的通信?
- 是否尽可能的局部通信?
- 通信操作是否能并行执行?
- 同步任务的计算能否并行执行?

内容概要

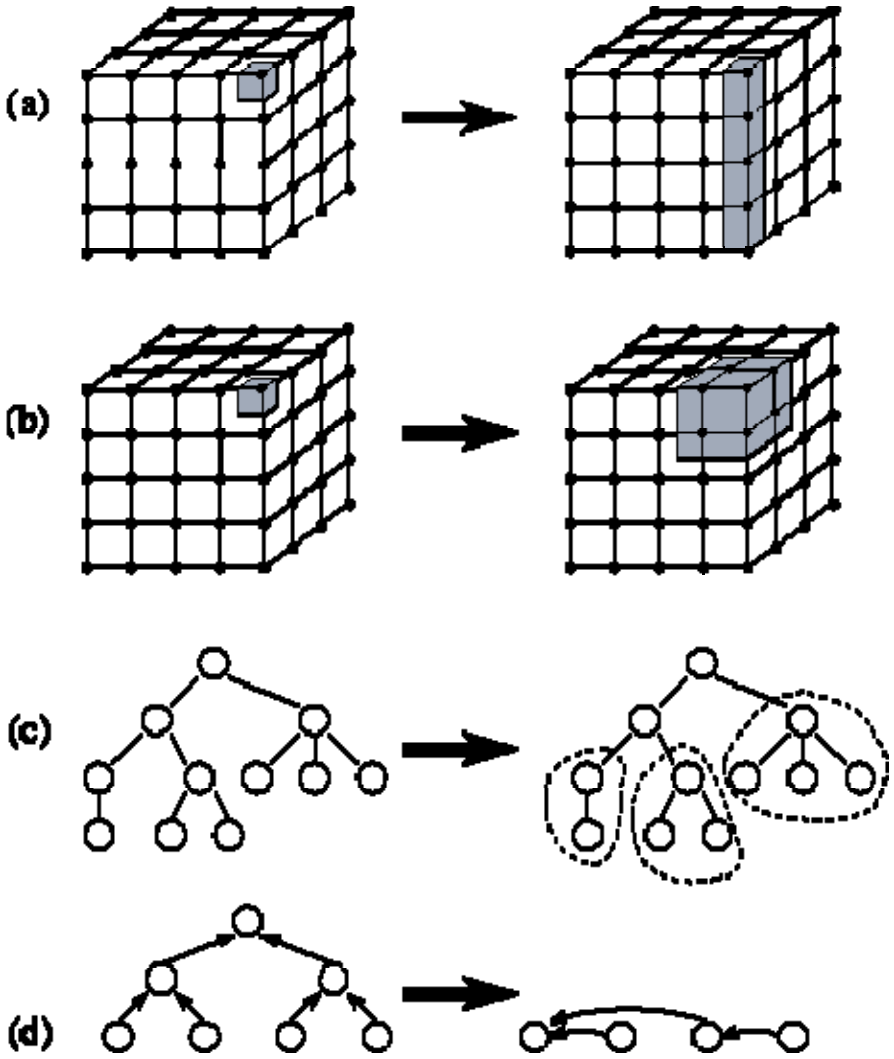
- 并行算法的基本设计技术
- 并行算法的一般设计过程
  - 划分
  - 通信
  - 组合
  - 映射

方法描述

- 组合是由抽象到具体的过程，是使得组合的任务能在一类并行机上有效地执行
- 合并小尺寸任务，减少任务数。如果任务数恰好等于处理器数，则也完成了映射过程
- 通过增加任务的粒度和重复计算，可以减少通信成本
- 保持映射和扩展的灵活性，降低软件工程成本

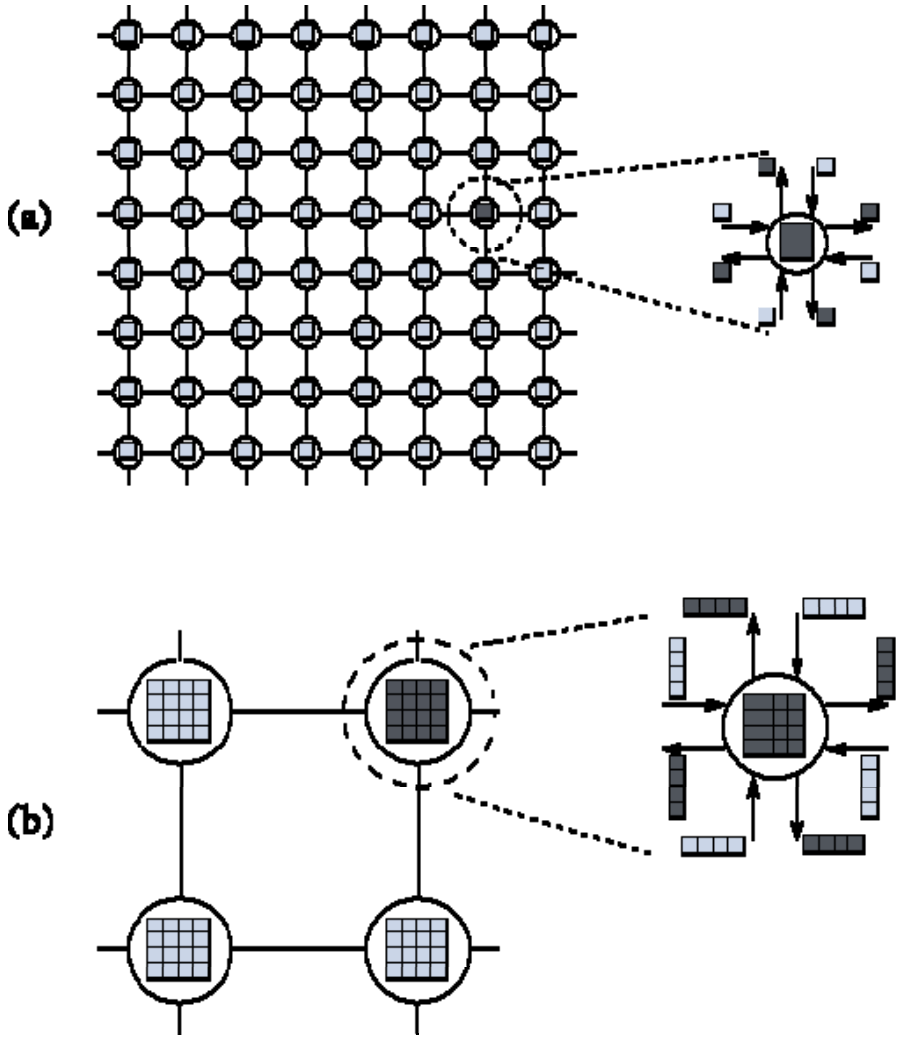
组合例子

- 图(a)，将分解维度从 3->2，子任务的规模变大
- 图 (b)，将相邻的子任务合并，形成高粒度的分解
- 图(c)，合并分治结构下的子树
- 图(d)，合并树算法中的节点



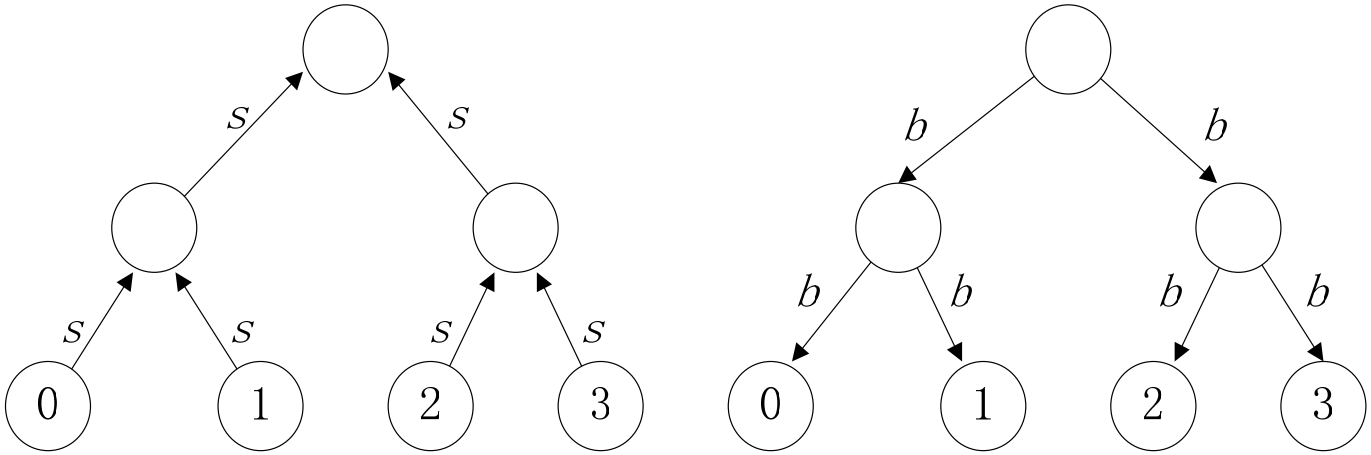
表面-容积效应
（Surface-to-Volume Effects）

- 通信量与任务子集的表面成正比，计算量与任务子集的体积成正比
- 增加重复计算有可能减少通信量



重复计算
（Replication Computation）

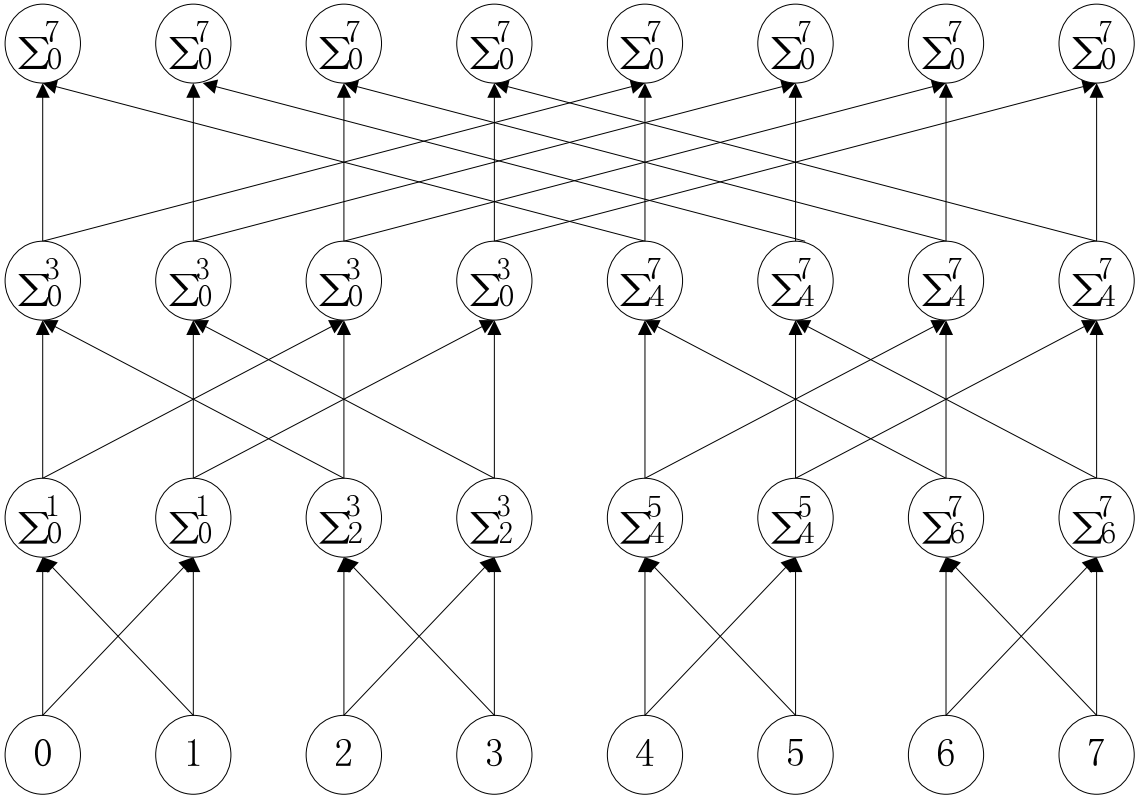
- 示例：二叉树上N个处理器求N个数的全和，要求每个处理器均保持全和



二叉树上求和，共需2logN步

重复计算（2）

- 示例：二叉树上N个处理器求N个数的全和，要求每个处理器均保持全和。



蝶式结构求和，使用了重复计算，共需logN步

重复计算（3）

- 重复计算减少通信量，但增加了计算量，应保持恰当的平衡
- 重复计算的目标应减少算法的总运算时间

组合判据

- 增加粒度是否减少了通信成本?
- 重复计算是否已权衡了其得益?
- 是否保持了灵活性和可扩展性?
- 组合的任务数是否与问题尺寸成比例?
- 是否保持了类似的计算和通信?
- 有没有减少并行执行的机会?

内容概要

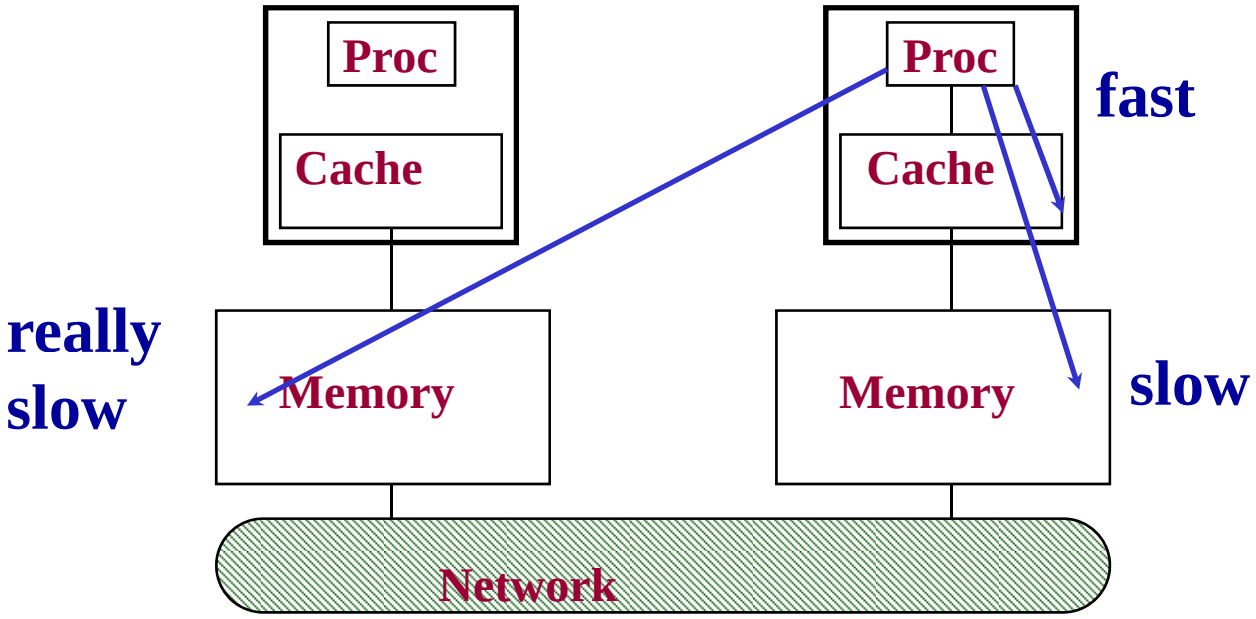
- 并行算法的基本设计技术
- 并行算法的一般设计过程
  - 划分
  - 通信
  - 组合
  - 映射

方法描述

- 每个任务要映射到具体的处理器，定位到运行机器上
- 任务数大于处理器数时，存在负载平衡和任务调度问题
- 映射的目标：减少算法的执行时间
  - 并发的任务 → 不同的处理器
  - 任务之间存在高通信的 → 同一处理器
- 映射实际是一种权衡，属于**NP**完全问题

映射的策略

- 两种策略
  - 使得任务可以被不同的处理器并发地执行，增强并发性（**concurrency**）
  - 将通信频繁的任务放到同一个处理器上，增强局部性（**locality**）

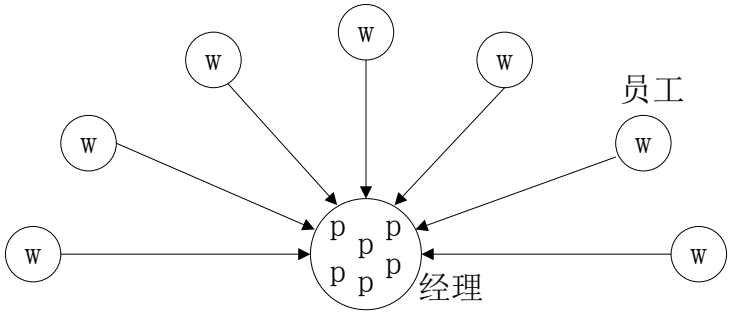


负载平衡算法

- 静态的：事先确定
- 概率的：随机确定
- 动态的：执行期间动态负载
- 基于域分解的：
  - 递归对剖
  - 局部算法
  - 概率方法
  - 循环映射

任务调度算法

- 任务放在集中的或分散的任务池中，使用任务调度算法将池中的任务分配给特定的处理器。下面是两种常用调度模式：
- 经理/雇员模式



- 非集中模式：分布式调度，每个处理器维持一个任务池

映射判据

- 采用集中式负载平衡方案，是否存在通信瓶颈？
- 采用动态负载平衡方案，调度策略的成本如何？

小结

- 将给定任务划分为小的任务，可用域分解或功能分解法
- 分析任务间的通信需求
- 使用组合方法，在保持灵活性的同时，减少通信和开发成本
- 将任务分配给处理器，使用负载均衡和任务调度技术提高映射的质量，目标是为了最小化总的执行时间

课程小结

- 并行算法的基本设计技术
  - 平衡树方法：求最大值、计算前缀和
  - 倍增技术：表序问题、求森林的根
  - 分治策略：串排序算法
  - 划分原理：均匀划分（**PSRS**排序）、对数划分（并行归并排序）、方根划分（**Valiant**归并排序）、功能划分（ **(m,n)-**选择）
  - 流水线技术：五点的**DFT**计算
- 并行算法的一般设计过程
  - 划分：域分解和功能分解
  - 通信：任务间的数据交换
  - 组合：任务的合并使得算法更有效
  - 映射：将任务分配到处理器，并保持负载平衡

推荐网站和读物

- 《并行计算》
  - 第6章：并行算法的基本设计技术
  - 第7章：并行算法的一般设计过程

- **Ian Foster, Designing and Building Parallel Programs**
  - **http://www-unix.mcs.anl.gov/dbpp/**

下一讲

- 稠密矩阵运算
  - 《并行计算－结构、算法、编程》第9章
- 并行程序设计基础
  - 《并行计算－结构、算法、编程》第12章