高性能计算与云计算

第四讲 并行算法设计（1）

胡金龙，董守斌

{jlhu, sbdong}@scut.edu.cn

华南理工大学计算机学院

广东省计算机网络重点实验室

Communication & Computer Network Laboratory (CCNL)

内容概要

- 并行算法的一般概念
- 并行计算模型
  - **PRAM模型**
  - **BSP模型**
  - **logP模型**
- 并行算法的一般设计方法

并行算法定义和分类

- **并行算法**

  一组可同时执行且可互相协作的诸进程的集合

- **分类**

  数值算法：　基于代数等数学运算
  非数值算法：　基于排序、选择、搜索 、匹配等符号处理

  同步算法：　进程执行需要相互等待
  异步算法：　各进程可以独立执行

  分布算法
  局网环境下
  网络计算：工作站机群　*COW*
  元计算 / 网格计算：　*Internet* 环境下

  确定算法：　求解过程为确定的步骤　以及算法结果 / 时间复杂性确定的
  非确定算法：　如：随机算法、智能算　法等

并行算法的表示

- **par-do**语句

```
for i=1 to n par-do        或    for i=1 to n do in parallel
     .                                  .
     .                                  .
     .                                  .
end for                          end for
```

- **for all**语句

```
for all Pi, where 0≤i≤k do
     .
     .
     .
end for
```

并行算法的复杂性度量（1）

- 运行时间 $t(n)$
- 处理器数目 $p(n)$
- 成本 $c(n)$： $c(n)=t(n)\times p(n)$

n: 问题规模

- 成本最优性：若 $c(n)$ 等于在最坏情形下串行算法所需要的时间（**Worst-Case Complexity**），则并行算法是成本最优（**Cost Optimal**）的。

并行算法的复杂性度量（2）

- 加速比$S_p(n)$：$S_p(n)=t_s(n)/t_p(n)$，其中$t_s(n)$为求解问题的最快的串行算法在最坏情形下所需的运行时间，$t_p(n)$为求解同一问题的并行算法在最坏情形下的运行时间
  - 加速比$S_p(n)$反映算法的并行性对运行时间的改进程度
  - 若$S_p(n)=p(n)$，则为线性加速；若$S_p(n)>p(n)$，则为超线性加速
- 并行效率$E_p(n)$：$E_p(n)=S_p(n)/p(n)$， $0<E_p(n)<=1$
  - 反映了并行系统中处理器的利用程度
- 工作量（或运算量）$W(n)$：并行算法所执行的总操作步数（与处理器的数目无关）

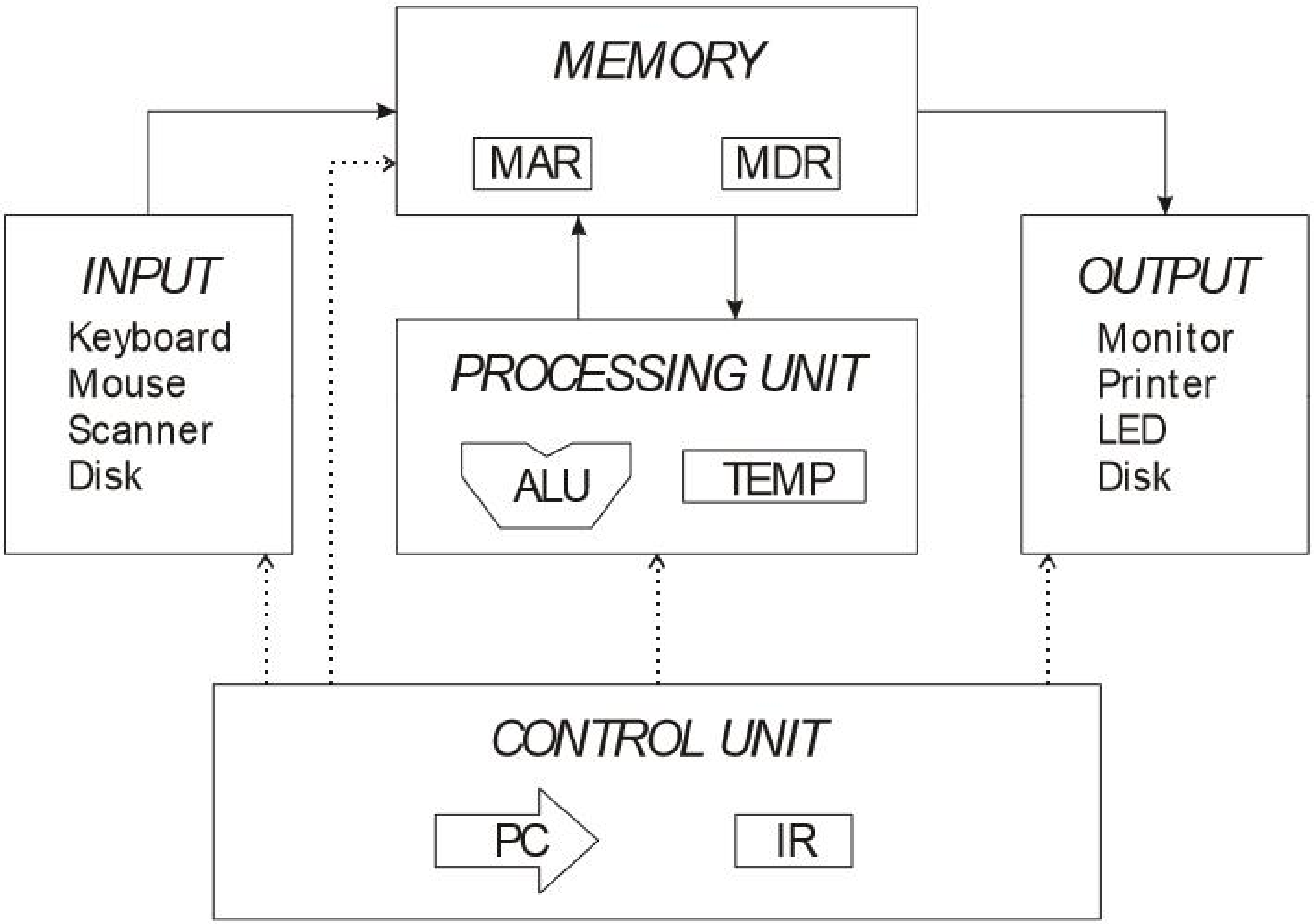
内容概要

- 并行算法的一般概念
- 并行计算模型
  - **PRAM模型**
  - **BSP模型**
  - **logP模型**
- 并行算法的一般设计方法

并行计算模型

- 并行计算模型：对并行计算机的抽象，为设计、分析和评价算法提供基础
- 为什么重要？
  - 串行计算机：冯诺依曼（von Neumann）计算模型
  - 并行计算机？
- 目标：在软件和硬件之间的一个桥梁
  - 可用于算法开发的抽象体系结构
  - 可准确反映目前机器的局限性
  - 可用于检查算法的性能（无需编程）

Von Neumann Model

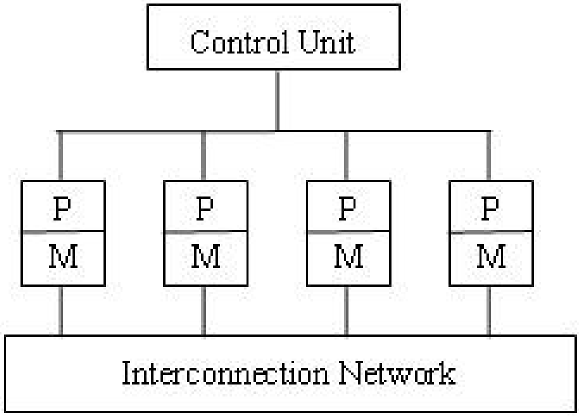


并行计算模型

- 对并行计算模型的要求：
  - 简单化：使得复杂问题可分析
  - 细节化：可用以发现系统瓶颈
- 主要模型
  - **PRAM模型**
  - **BSP模型**
  - **LogP模型**

各种专用和通用的并行计算模型

- **SISD, MIMD**
- **SIMD-DM模型**
  - ➢ **SIMD-LC (Linear Connected) 线性连接**
  - ➢ **SIMD-MC (Mesh Connected) 网孔连接**
  - ➢ **SIMD-TC (Tree Connected)树形连接**
  - ➢ **SIMD-MT (Mesh Tree)树网连接**
  - ➢ **SIMD-HC (Hypercube Connected)超立方连接**
  - ➢ **SIMD-CCC (Cube Connected-Cycles) 立方环连接**
  - ➢ **SIMD-SE (Shuffle Exchange)洗牌交换连接**
  - ➢ **SIMD-BF (Butterfly) 碟形网连接**



- **PRAM模型（SIMD-SM模型）**
  - ➢ **PRAM-CRCW，PRAM-CREW，PRAM-EREW**
  - ➢ **APRAM**
- **BSP模型(一种MIMD-DM)**
- **LogP模型(基于MPC)**

算法在不同计算模型上可能呈现不同的计算复杂度

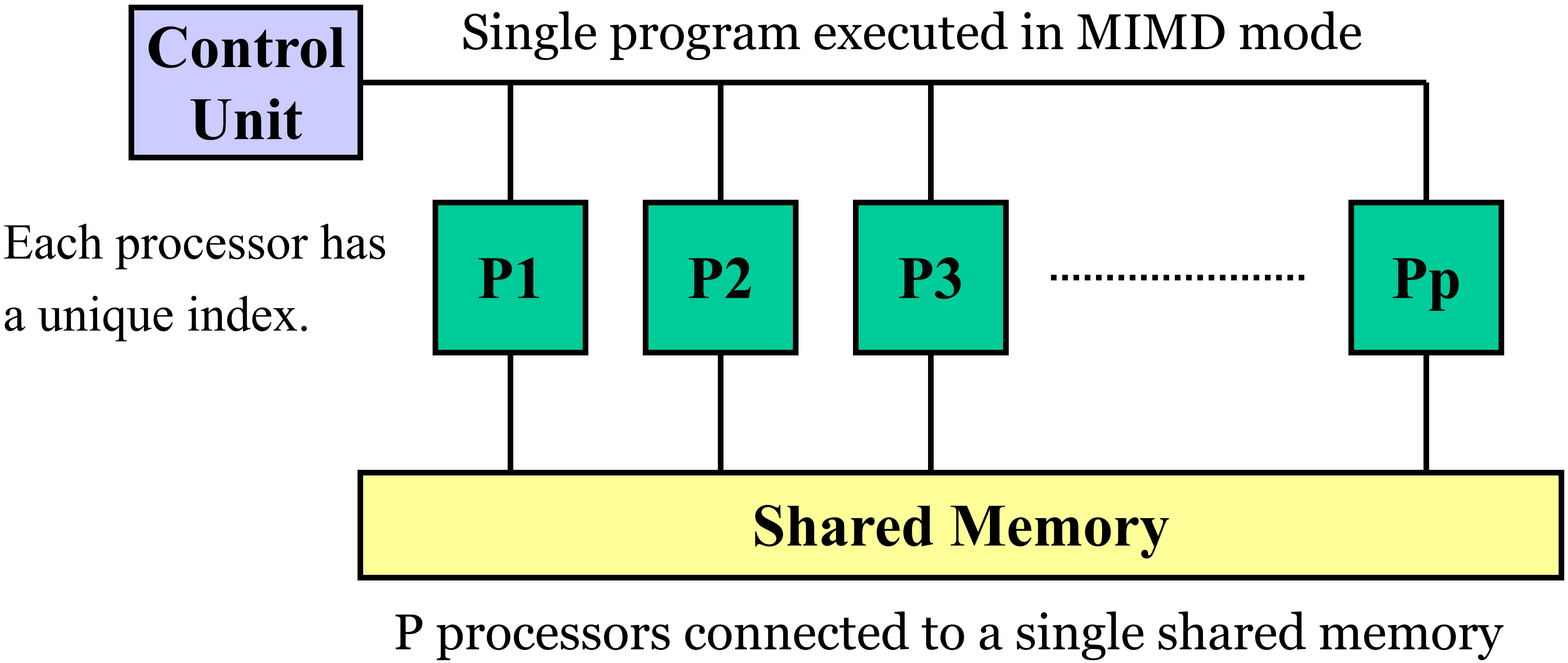
内容概要

- 并行算法的一般概念
- 并行计算模型
  - PRAM模型
  - BSP模型
  - logP模型
- 并行算法的一般设计方法

PRAM模型

- **PRAM (Parallel Random Access Machine)**
  - 由Fortune和Wyllie1978年提出，又称SIMD-SM模型
  - 有一个集中的共享存储器和一个指令控制器，通过共享存储（SM）的Read/Write交换数据，隐式同步计算
  - 在一个时钟周期内，每个处理器执行一条指令，可完成3个操作
    - ✓从存储器取出操作数
    - ✓完成一个算术、逻辑运算
    - ✓将结果存回存储器

PRAM结构图


Control Unit
Single program executed in MIMD mode
Each processor has a unique index.
P1
P2
P3
Pp
Shared Memory
P processors connected to a single shared memory

PRAM模型特点

- 全局共享存储，单一地址空间
- 同步、通信和并行化的开销为零
- 优点
  - 适合并行算法表示和复杂性分析，易于使用
  - 隐藏了并行机的通讯、同步等细节
- 缺点
  - 不适合MIMD并行机（Why？），忽略了SM的竞争、通讯延迟等因素

存储数据的存取模式（1）

		Reads from same location	
		Exclusive	Concurrent
Writes to same location	Exclusive	EREW Least “powerful”, most “realistic”	CREW Default
	Concurrent	ERCW Not useful	CRCW Most “powerful”, Requires conflict resolution schema

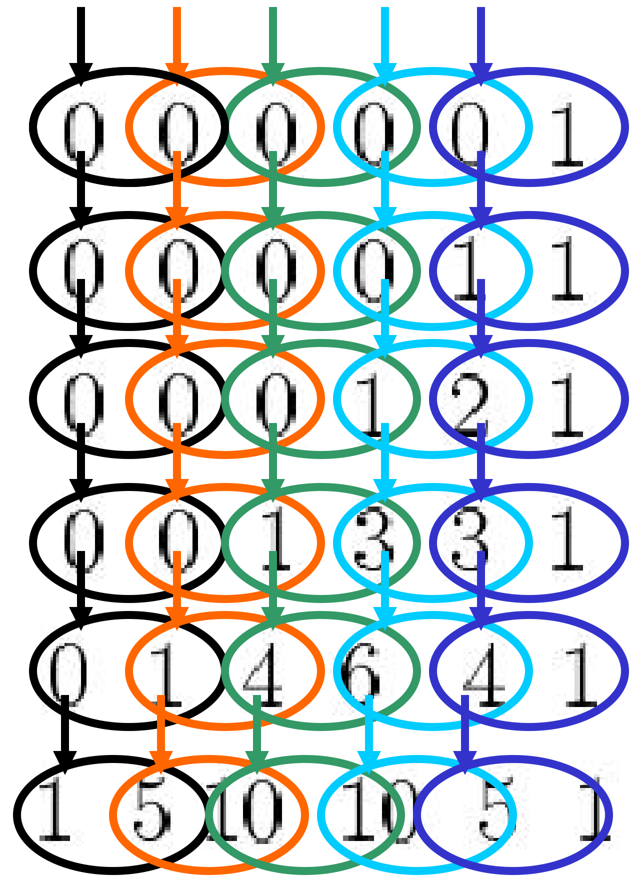
存储数据的存取模式（2）

- **CRCW：并发读并发写，冲突解决模式：**
  - 相同/公共并发读写（**Common CRCW**）：仅允许写入相同数据
  - 优先并发读写（**Priority CRCW**）：仅允许优先级最高的处理器写入
  - 任意并发读写（**Arbitrary CRCW**）：允许任意处理器自由写入
- **CREW：并发读互斥写**
- **EREW：互斥读互斥写**

例子：CREW-PRAM

- 假设初始，表 **A** 为 [0,0,0,0,0,1]，有如下的并行程序

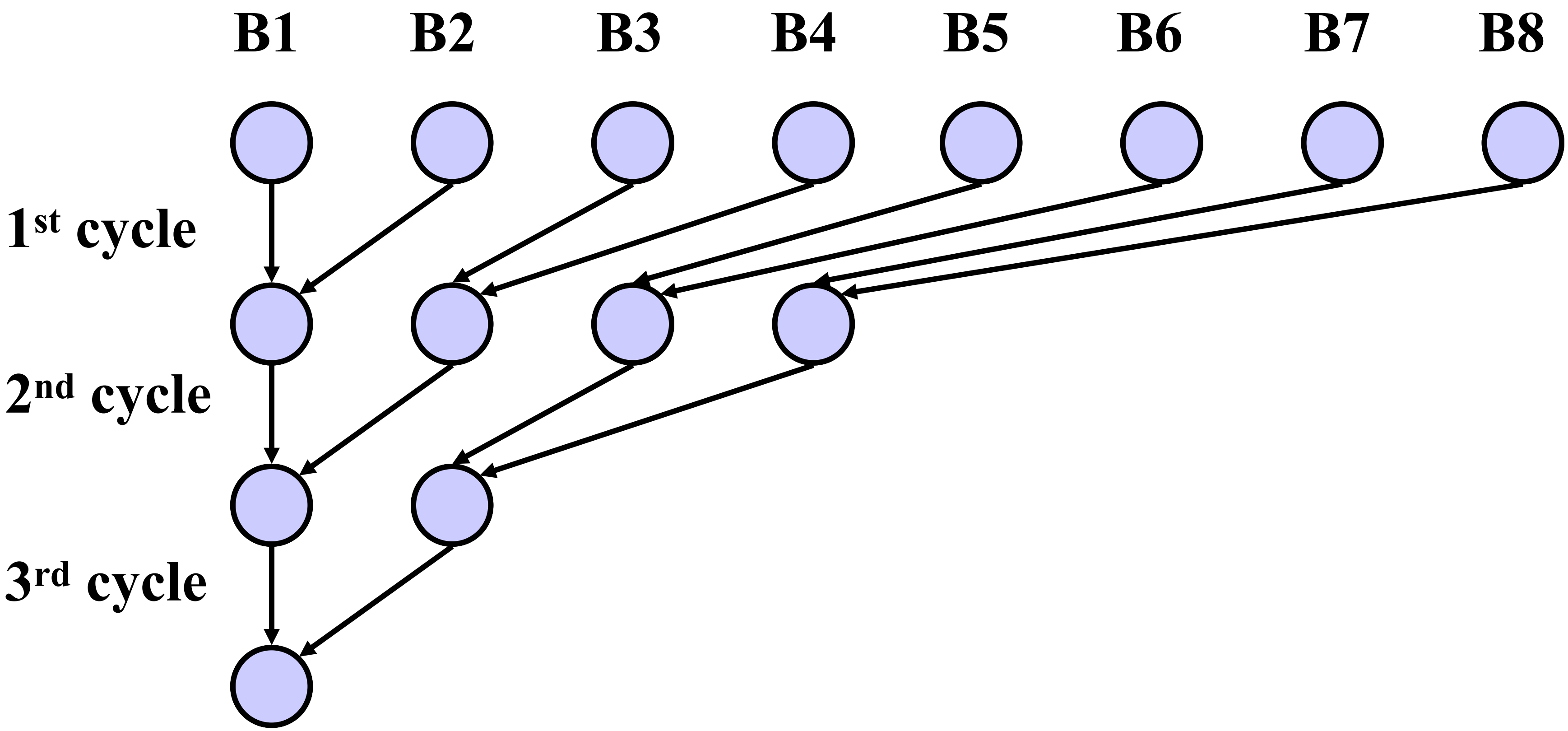
for each $1 \le i \le 5$ do in parallel
$A[i] := A[i] + A[i+1]$



Pascal Triangle

例子：归约（Reduction）

- 输入： Sequence x[1…*n*] of numbers (where $n = 2^h$)
- 输出： S = sum of numbers



例子：归约（2）

```
int B[1…n], X[1…n];

forall i = 1 . . n do
        B[ i ] = X[ i ];

for j = 1 . . h do
        forall i = 1 . . n / 2^j do
                B[ i ] = B[ 2 * i – 1 ] + B[ 2 * i ];
S = B[ 1 ];
```

- p个处理器的EREW机器上的时间复杂度？(如果 $n = p$)
  - ➢ O(log n)

例子：点积

- p 个处理器的EREW PRAM的机器上，做N维的矢量A和B的点积（inner product）
  - Step 1
    - 每个处理器：$2N/p$ 个加法和乘法
  - Step 2
    - 通过树归约方法计算p个局部和：$\log p$
  - 总的执行时间： $2N/p + \log p$
  - 与串行计算时间相比（加速比）： p as $N >> p$

计算能力比较

- **PRAM-CRCW是最强的计算模型**

$$T_{EREW} \geq T_{CREW} \geq T_{CRCW}$$

- 一个具有时间复杂度为$\mathbf{T_{CREW}}$或者$\mathbf{T_{CRCW}}$的算法，在**PRAM-EREW**模型上要花费**log*p***倍的时间去模拟实现

$$T_{EREW} = O(T_{CREW} \cdot \log p) = O(T_{CRCW} \cdot \log p)$$

其他PRAM

- 分相PRAM（Phased PRAM ）
  - 一个异步（Asynchronous）的PRAM模型
  - 各个处理器异步地执行局部程序，每个局部程序的最后一条语句是同步障指令
- 本地存储PRAM（Local memory PRAM）：LPRAM
  - 考虑本地存储和远程存储的开销不同
- 块PRAM（Block PRAM）：BPRAM
  - LPRAM + 通信开销

PRAM模型的优点

- 优点：
  - 简单：PRAM模型特别适合于并行算法的表达、分析和比较，使用简单
  - 易于扩展：根据需要，可以在PRAM模型中加入一些诸如同步和通信等需要考虑的内容
- 缺点：
  - 模型中使用了一个全局共享存储器，不适合于分布存储结构的并行机
  - PRAM模型是同步的，不能反映现实中很多系统的异步性
  - PRAM模型假设了每个处理器可在单位时间访问共享存储器的任一单元，因此要求处理机间通信无延迟、无限带宽和无开销，这是不现实的

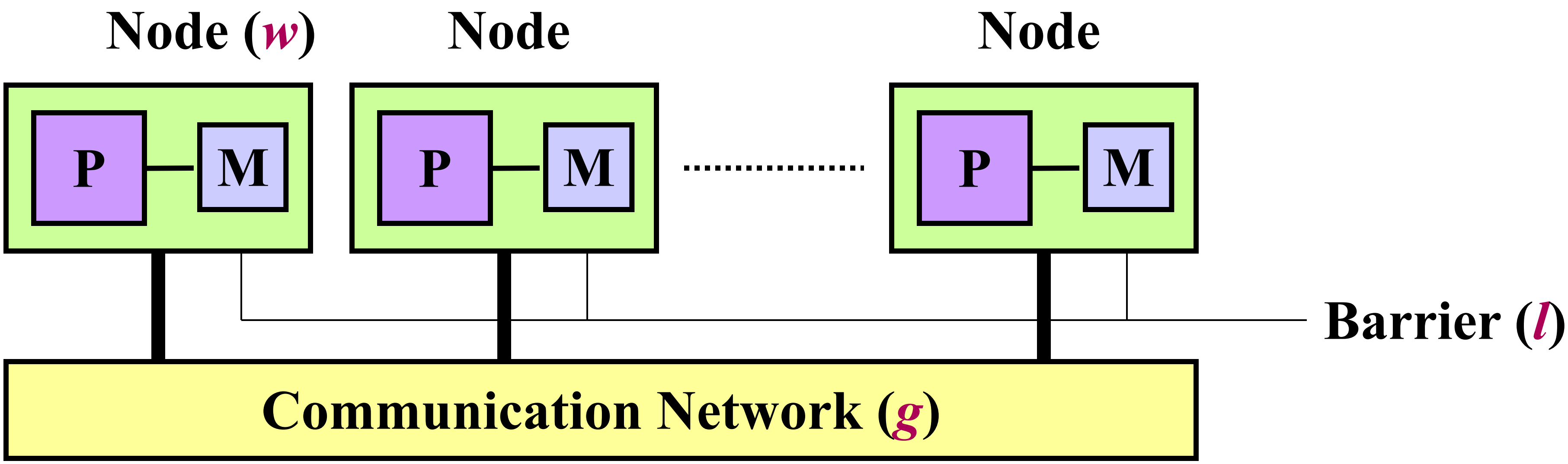
内容概要

- 并行算法的一般概念
- 并行计算模型
  - PRAM模型
  - BSP模型
  - logP模型
- 并行算法的一般设计方法

BSP模型

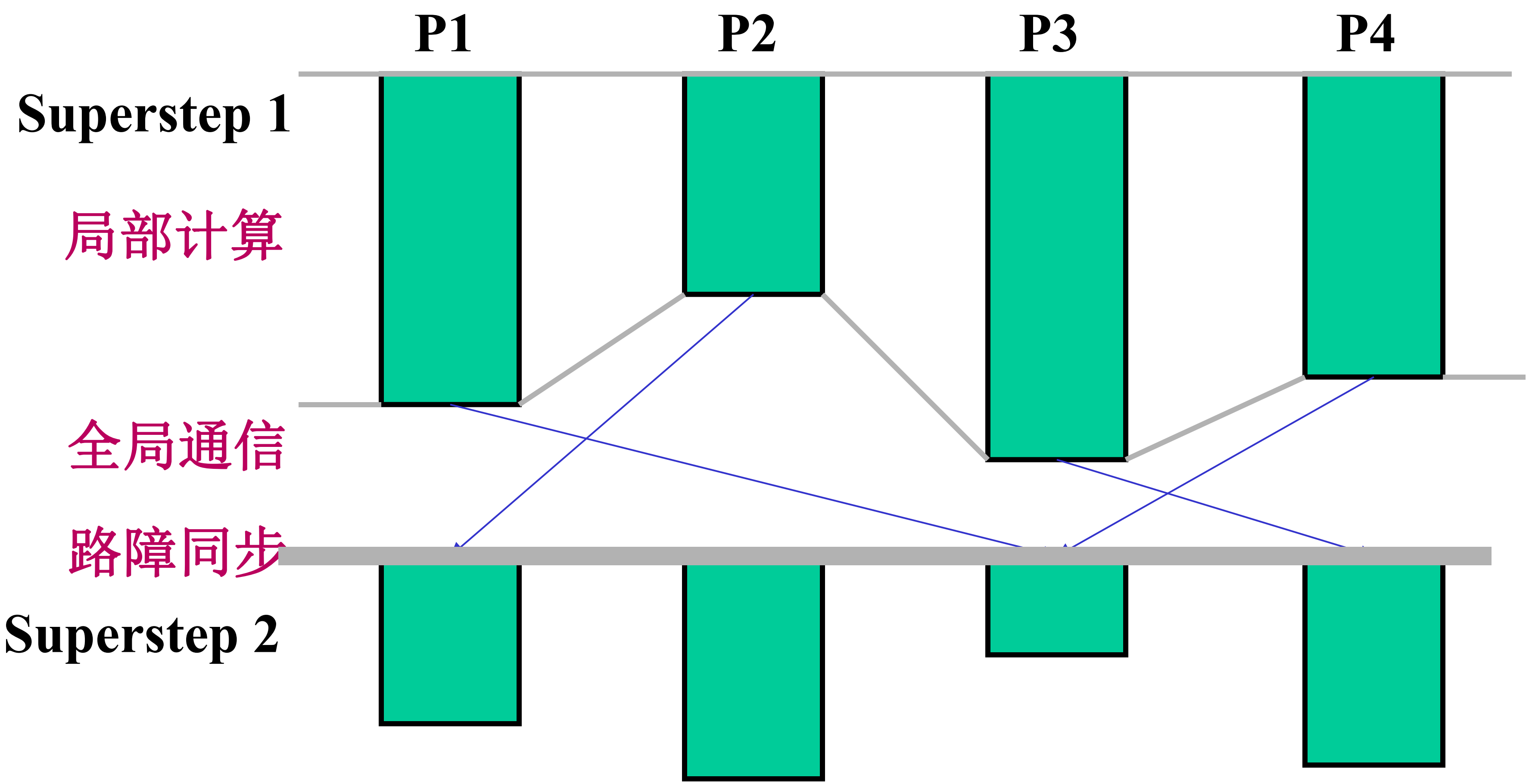
- 块同步模型（Bulk Synchronous Parallel）BSP
  - 是一种异步MIMD-DM（分布式存储）模型，支持消息传递系统，块内异步并行，块间显式同步
  - Harvard University 的Valiant于1990年提出，Oxford University的 McColl 在1993进行了修改
- 3 个组成部分
  - 节点（Node）：处理器，本地存储
  - 通信网络（Communication Network）：点到点，消息传递
  - 同步障（Barrier）：同步机制

BSP图例



超步

- **BSP计算过程由若干超级步（Super Step）组成**



模型参数

- 参数 *w*： 计算参数
  - 每个超步内的最大的计算时间
  - 计算操作最多为 *w* 个时钟周期（cycles）
- 参数 *g*： 带宽参数（time steps/packet）= 1/bandwidth
  - 单位消息所需的通信时钟数－网络带宽的倒数
  - 关系因子（relation coefficient） *h*: 每个节点至多发送和接收*h*个消息
  - 通信操作最多为 *gh* 个时钟周期
- 参数 *l*： 同步障参数（Barrier synchronization time）
  - 同步障操作最多为 *l* 个时钟周期

BSP时间复杂度

- **Valiant公式:** $\max\{w, gh, l\}$
  - 计算与通信重叠
- **McColl重写后的公式:** $w+gh+l$

例子：点积

- 用8个处理器进行点积计算
  - Superstep 1
    - ✓ 计算：本地和 $w = 2N/p$
    - ✓ 通信： h=1 (处理器 0,2,4,6 -> 1,3,5,7)
  - Superstep 2
    - ✓ 计算： 一个加法 $w = 1$
    - ✓ 通信： h=1 (处理器1,5 -> 3,7)
  - Superstep 3
    - ✓ 计算：一个加法 $w = 1$
    - ✓ 通信： h=1 (处理器 3 -> 7)
  - Superstep 4
    - ✓ 计算：一个加法 $w = 1$
  - 总的执行时间 = $2N/8 + 3g+3l+3$

例子：点积（2）

- 更一般地，对于p个处理器

  总的执行时间= $2N/p + \log p(g+l+1)$

  - $g \log p$：通信开销（communication overhead ）
  - $l \log p$：同步开销（ synchronization overhead）

- 对比 PRAM 模型
  - 总的执行时间= $2N/p + \log p$

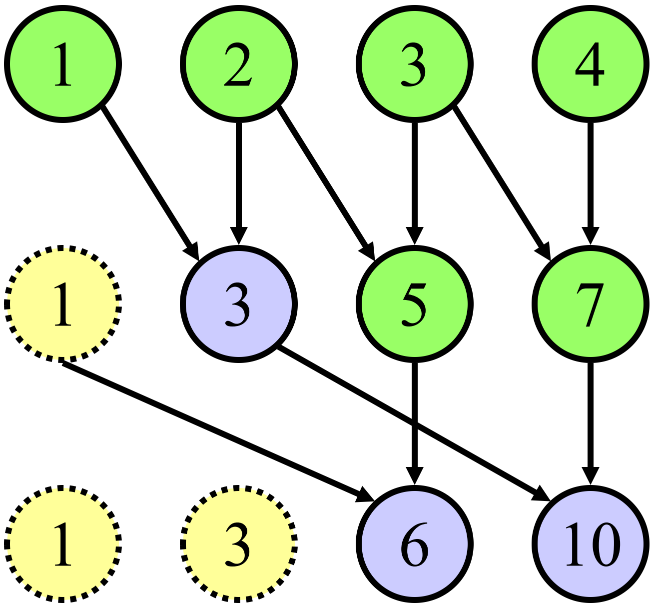
BSP程序库—BSPlib

- Initialization Functions
  - `bsp_init()`
    - Simulate dynamic processes
  - `bsp_begin()`
    - Start of SPMD code
  - `bsp_end()`
    - End of SPMD code
- Enquiry Functions
  - `bsp_pid()`
    - find my process id
  - `bsp_nprocs()`
    - number of processes
  - `bsp_time()`
    - local time
- Synchronization Functions
  - `bsp_sync()`
    - barrier synchronization
- DRMA Functions
  - `bsp_push_register()`
    - make region globally visible
  - `bsp_pop_register()`
    - remove global visibility
  - `bsp_put()`
    - push to remote memory
  - `bsp_get()`
    - pull from remote memory

http://www.bsp-worldwide.org/

BSP编程例子（1）

- 用BSPLib并行求4个整数1，2，3，4的前缀和（P171 习题5.10）
  - ⌈log p⌉ 个超步
  - 4 个处理器



P172 图 5.10

BSP编程例子（2）

```
int bsp_allsum(int x)
{
    int i, left, right;

    bsp_push_register(&right, sizeof(int));
    bsp_push_register(&left, sizeof(int));
    bsp_sync();

    right = x;
    for(i=1; i<bsp_nprocs(); i*=2) {
        bsp_put(bsp_pid()+i, &right, &left, 0, sizeof(int));
        bsp_sync();
        if(bsp_pid() >= i) right = left + right;
    }
    bsp_pop_register(&right);
    bsp_pop_register(&left);
    return right;
}
```

BSP模型的优缺点

- 优点：
  - Ø BSP模型试图为软件和硬件之间架起一座类似于冯诺伊曼机的桥梁，因此，BSP模型也常叫做桥模型
  - Ø 将处理器和路由器分开，强调了计算任务和通信任务的分开，而路由器仅仅完成点到点的消息传递，不提供组合、复制和广播等功能，这样做既掩盖具体的互连网络拓扑，又简化了通信协议

- 缺点：
  - Ø 需要显式同步，编程效率不高
  - Ø BSP模型中的全局障碍同步假定是用特殊的硬件支持的，这在很多并行机中可能没有相应的硬件

内容概要

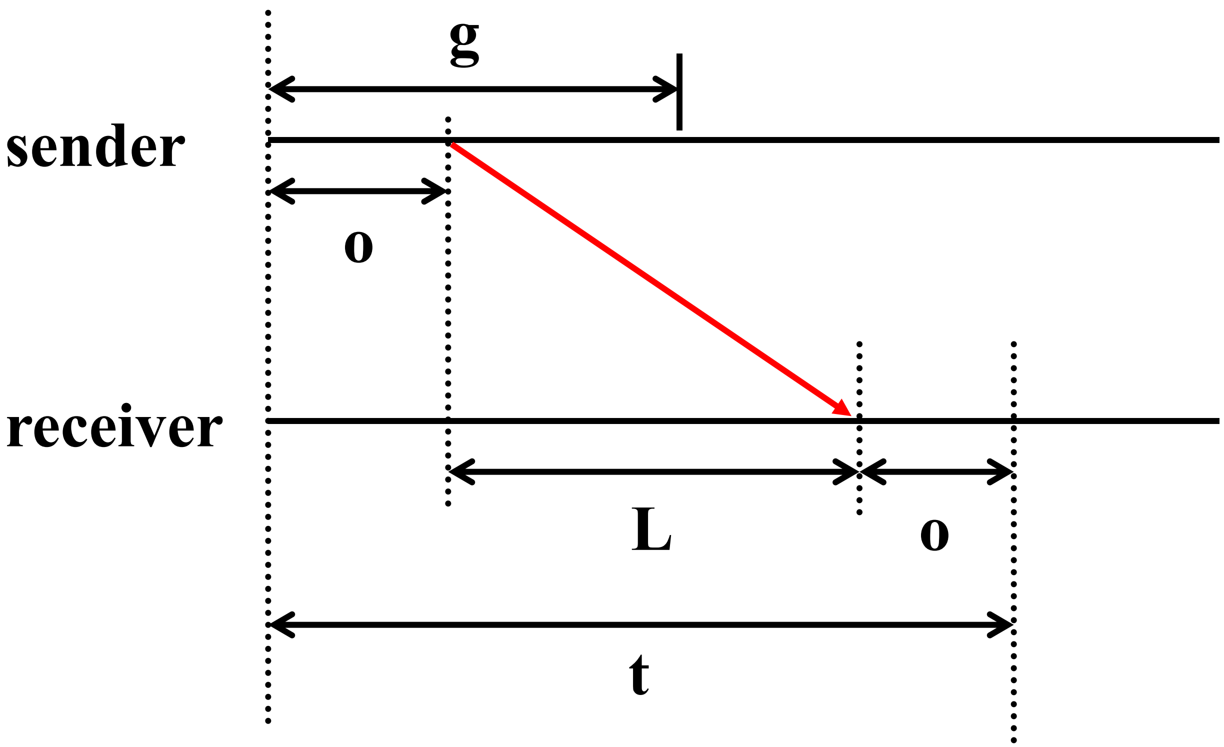
- 并行算法的一般概念
- 并行计算模型
  - PRAM模型
  - BSP模型
  - logP模型
- 并行算法的一般设计方法

LogP模型

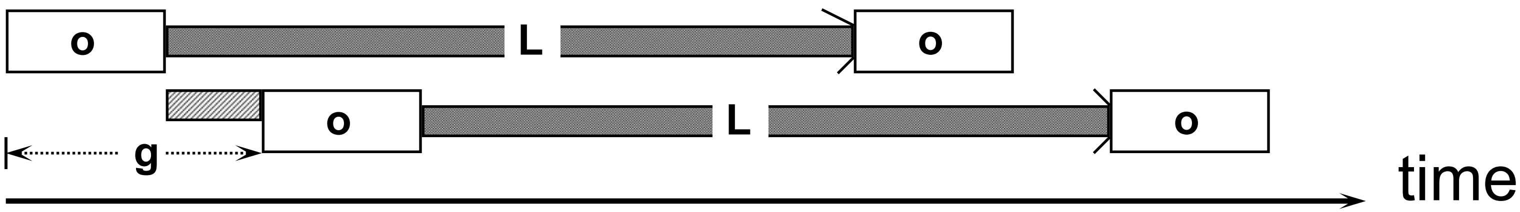
- 基本概念
  - 由Culler（1993）年提出的，是一种分布存储的、点到点通讯的多处理机模型，其中通讯由一组参数描述，实行隐式同步。
- 模型参数
  - L：网络时延（network latency）
  - o：通信开销（communication overhead）
  - g：gap=1/bandwidth
  - P：#processors

注：L和g反映了通讯网络的容量

LogP图例

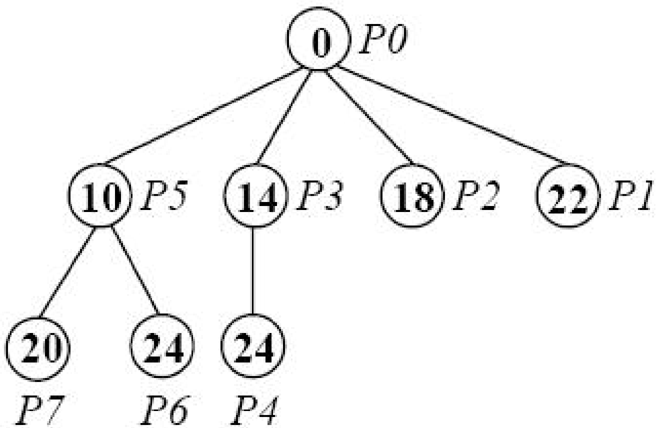


模型的使用



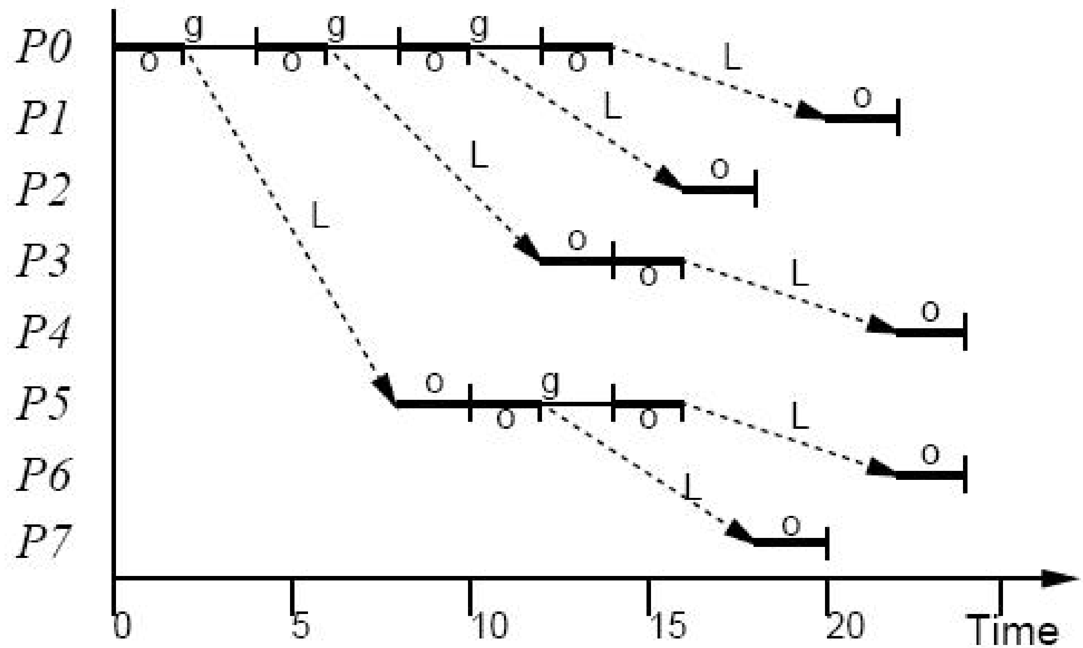
- 从处理器发送*n* 个消息到处理器，需要时间：**2o + L + g(n-1)**
- 从一个处理器广播单个数据包到其他*P*-1处理器，所需的最短时间是多少？
  - **LogP参数：** P= 8，L=6，g=4，o=2

例子：优化组播树
（Optimal Broadcast Tree）



LogP 参数：

P= 8， L=6， g=4， o=2



- 在时刻24接收到最后一个值

真实并行机的LogP参数

Machine	L (*us*)	o (*us*)	g (*us*)	P	t_0 (*us*)	r_a (MB/s)
IBM SP2	1.25	22.38	24	400	46	35
Meico CS-2	10	3.8	13.8	64	17.6	43

t_0: 通信时延（Communication Latency）

r_a :通信带宽（ Communication Bandwidth）

LogP模型的优点

- 异步（ Asynchronous ）模型，没有同步障
- 捕捉了并行计算机的通讯瓶颈
- 通信由一组参数描述，但并不涉及具体的网络结构，隐藏了并行机的网络拓扑、路由、协议
- 可以应用到共享存储、消息传递、数据并行的编程模型中

LogP模型的缺点

- 难以进行算法描述、设计和分析
- 对网络中的通信模式描述的不够深入。如重发消息可能占满带宽、中间路由器缓存饱和等未加描述
- LogP模型主要适用于消息传递算法设计，对于共享存储模式，则简单地认为远地读操作相当于两次消息传递，未考虑流水线预取技术、Cache引起的数据不一致性以及Cache命中率对计算的影响

BSP和LogP的比较

- **BSP分为：BSP块同步，BSP子集同步，BSP进程对同步**
- **BSP进程对同步＝LogP**
- **BSP可以常数因子模拟LogP，LogP可以对数因子模拟BSP**
- **BSP＝LogP＋Barriers－Overhead**
- **BSP提供了更方便的程序设计环境，LogP更好地利用了机器资源**
- **BSP似乎更简单、方便和符合结构化编程**

三种模型的比较（表5.1）

模型 特性	PRAM	BSP	logP
体系结构	SIMD-SM	MIMD-DM	MIMD-DM
计算模型	Synchronous	Asynchronous	Asynchronous
同步机制	Automatic	Barrier	Implicated
参数	l	p,g,l	l,o,g,p
粒度	Fine/Moderate	Moderate /Coarse	Moderate /Coarse
通信	Shared Variable	Message Passing	Message Passing
地址空间	Global	Single/Multiple	Single/Multiple

内容概要

- 并行算法的一般概念
- 并行计算模型
  - PRAM模型
  - BSP模型
  - logP模型
- 并行算法的一般设计方法

并行算法的一般设计方法

- 串行算法的直接并行化
- 从问题描述开始设计并行算法
- 借用已有算法求解新问题

串行算法的直接并行化

- 方法描述
  - 发掘和利用现有串行算法中的并行性，直接将串行算法改造为并行算法
- 评注
  - 由串行算法直接并行化的方法是并行算法设计的最常用方法之一
  - 不是所有的串行算法都可以直接并行化的
  - 一个好的串行算法并不能并行化为一个好的并行算法
  - 许多数值串行算法可以并行化为有效的数值并行算法

例子：快排序（Quicksort）算法

- 问题描述：将给定序列($A_1, \ldots, A_n$)变成一个有序序列
- 计算原理：
  - 基于分治策略的递归排序：将一个序列分成两个非空子系列，前一个子序列的任一元素都小于后一个子序列的任何元素。对两个子序列进行递归调用，直至子系列只有两个元素为止。

串行快排序算法

Procedure QUICKSORT(A, q, r)
//输入无序序列($A_q,\ldots,A_r$)
//输出有序序列($A_q,\ldots,A_r$)
begin
    if $q<r$ then
        (1) $x= A_q$
        (2) $s=q$
        (3) for $i=q+1$ to r do
                if $A_i \le x$ then
                    (i) $s=s+1$
                    (ii) swap(A_s, A_i)
                end if
        (4) swap(A_q, A_s)
        (5) QUICKSORT(A, q, s)
        (6) QUICKSORT(A, s+1, r)
End

算法**6.1**

算法复杂度?

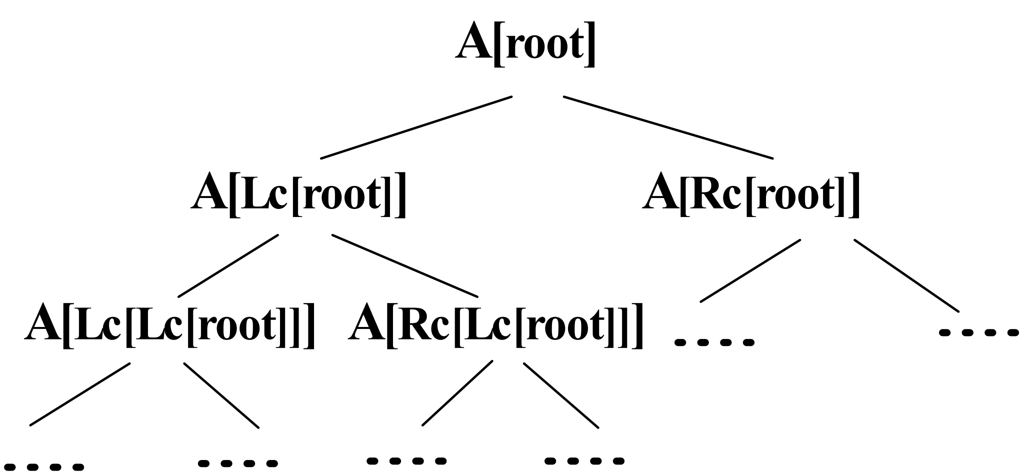
这个算法如何并行化?

并行快排序算法描述

- 将快排序变成构造一个二叉树，其中主元是根，小于等于主元的元素处于左子树，大于主元的元素处于右子树
- 共享变量root，LC[1..n]，RC[1..n]，及待排序数据A[1..n]
- n个处理器，P_i存有A[i]，f_i中保存期元素是主元的处理器号
- 问题描述转化为：

  输入：序列(A1,…,An)和n个处理器

  输出：供排序用的一棵二叉排序树
- 得到二叉排序树后，只要中序遍历即可得到排序序列



并行快排序算法

算法6.2 PRAM-CRCW

Begin

(1) for each processor i **do**

(1.1) root=i　//P_i将处理器号i并发写入SM变量root，root的值是不确定的

(1.2) f_i=root　//P_i并发读入root到LM变量f_i中

(1.3) $LC_i=RC_i=n+1$　//LC_i和RC_i初始化，使得不指向任何处理器

end for

(2) repeat for each processor i<>root **do**　//A_i是LM变量, A_{fi}是SM变量;

if $(A_i<A_{fi}) \vee (A_i=A_{fi} \wedge i<f_i)$ then　($A_i= A_{fi}$ and $i<f_i$)为了排序稳定

(2.1) LC_{fi}=i　//P_i将i并发写入SM变量LC_{fi}, 竞争为f_i的左孩子

(2.2) **if** i=LC_{fi} **then** exit **else** $f_i=LC_{fi}$ **endif**

else

(2.3) RC_{fi}=i　//P_i将i并发写入SM变量RC_{fi}, 竞争为f_i的右孩子

(2.4) **if** i=RC_{fi} **then** exit **else** $f_i=RC_{fi}$ **endif**

endif

end repeat

End

时间分析：运行时间 $T(n)=O(\log n)$, 处理器数 $p(n)=n$, 并行算法的成本 $c(n)=O(n\log n)$

并行算法的一般设计方法

- 串行算法的直接并行化
- 从问题描述开始设计并行算法
- 借用已有算法求解新问题

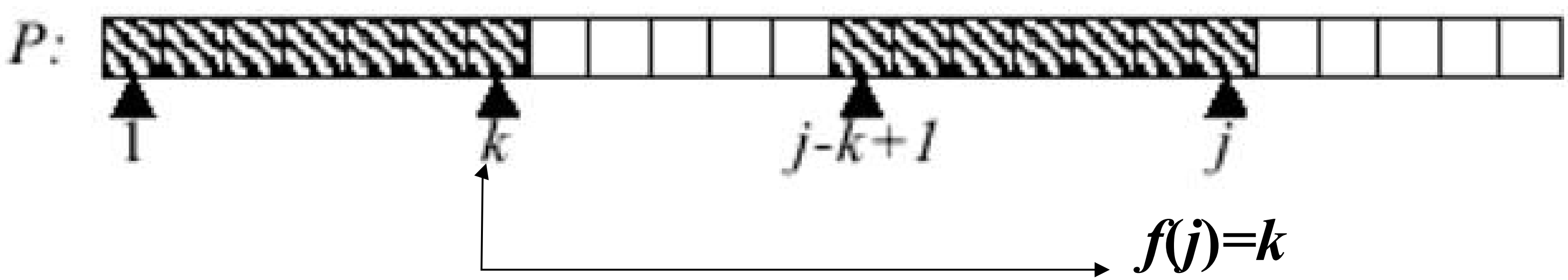
从问题描述开始设计并行算法

- 方法描述
  - 从问题本身描述出发，不考虑相应的串行算法，设计一个全新的并行算法
- 评注
  - 挖掘问题的固有特性与并行的关系
  - 设计全新的并行算法是一个挑战性和创造性的工作
  - 范例：利用串的周期性的**Galil**和**Vishkin**算法（并行串匹配算法）

例子：串匹配算法

- **串匹配（String Matching）问题描述：给定长度为n的正文串text和长度为m的模式串pat（$m \le n$），找出pat在text中出现的所有位置i**
- **计算方法：**
  - 一般方法：将text分成$n-m+1$个长度为m的子串，检查是否与pat相匹配，时间复杂度$(n-m+1)m=O(nm)$
  - KMP方法（算法6.4），线性复杂度$O(n+m)$。如何做到？

失效函数（Failure Function）



- 失效函数 $f(j)$=largest $k < j$ such that $P_{1,k}=P_{j-k+1,j}$
- $f(j)=0$ if no such k

j	1	2	3	4	5	6	7	8	9	10	11	12
P:	A	T	C	A	C	A	T	C	A	T	C	A
f:	0	1	0	1	0	1	2	3	4	2	3	4

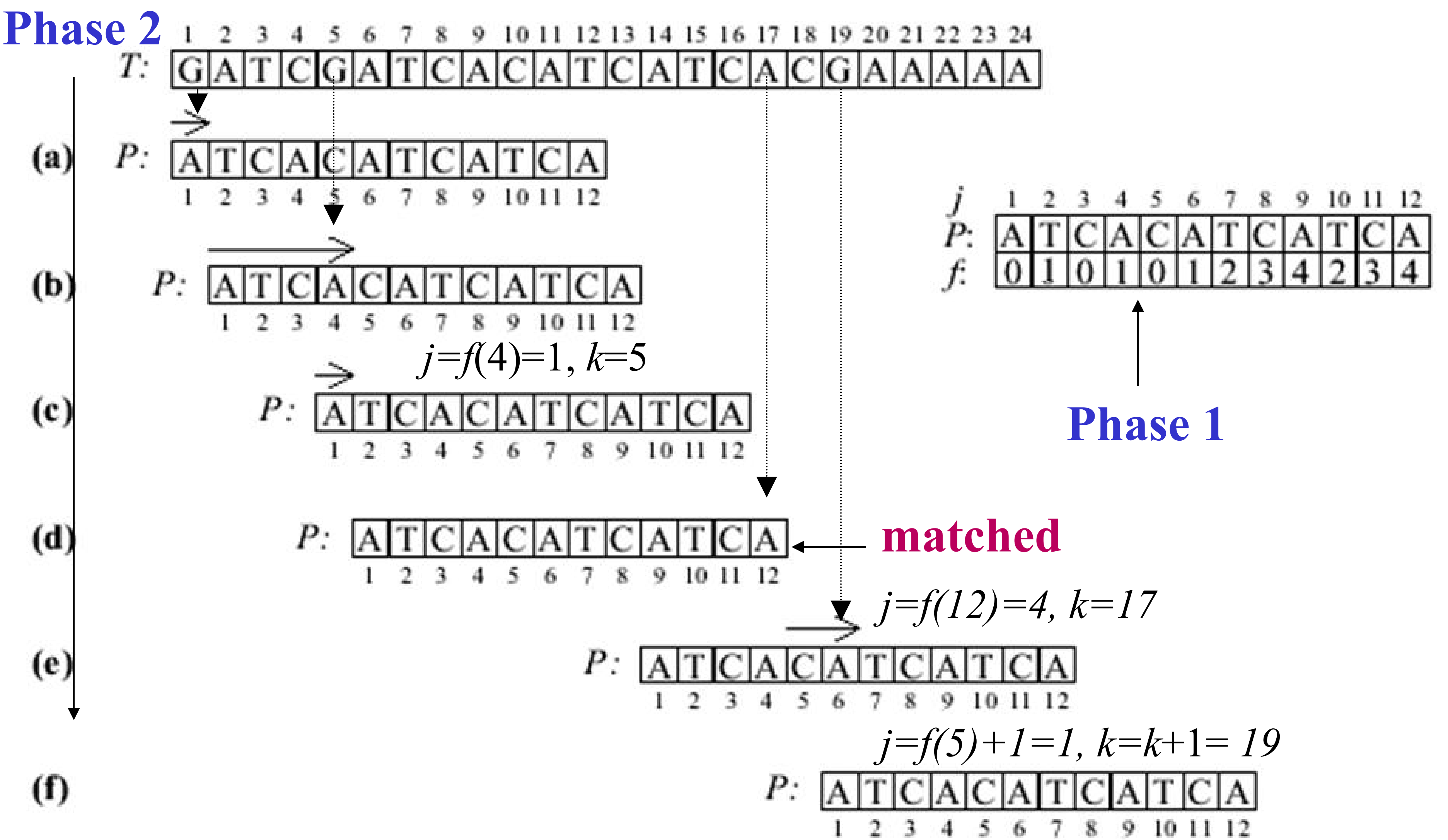
算法6.3　失效函数的计算方法

KMP串匹配算法

```
输入：T[1:n], P[1:m], f[1:m+1]
输出：match(i)
Bejin
    (1) j = 1; k = 1
    (2) while k-j ≤ n-m do
            (2.1) if T(k) = P(j) then
                    (i) k = k+1
                    (ii) j = j+1
                    (iii) if j = m+1 then match(k-m) = 1 endif
                  endif
            (2.2) if T(k)≠P(j) then
                  (i) j = f(j)
                  (ii) if j = 0 then
                         k = k+1;
                         j = 1 endif
                  endif
        endwhile
End
```

算法6.4

KMP算法的例子



Phase 2
T: G A T C G A T C A C A T C A T C A C G A A A A A
(a) P: A T C A C A T C A T C A
(b) P: A T C A C A T C A T C A
j=f(4)=1, k=5
(c) P: A T C A C A T C A T C A
(d) P: A T C A C A T C A T C A
matched
j=f(12)=4, k=17
(e) P: A T C A C A T C A T C A
j=f(5)+1=1, k=k+1= 19
(f) P: A T C A C A T C A T C A
j 1 2 3 4 5 6 7 8 9 10 11 12
P: A T C A C A T C A T C A
f: 0 1 0 1 0 1 2 3 4 2 3 4
Phase 1

并行串匹配算法

- KMP算法的精髓在于使用了失效函数，使用它来调整两个指针j和k。
- KMP算法不容易并行化（why？）
  - 指针来回移动
  - T串分段并行处理

并行串匹配算法

- 将串分为（n / m -1）个长度为 m – 1的部分。前向搜索前缀，逆向搜索后缀。如果前缀和后缀的长度为m，而且没有间隙，则找到pattern

  例子：text＝“abcabcabcabc”, pat＝“abcab”

Text	a	b	c	a	b	c	a	b	c	a	b	c
Part	T_1				T_2				T_3			
ϱ	T	F	F	T	F	F	T	F	—	—	—	—
φ	—	—	—	—	T	F	F	T	F	F	T	F
$MATCH$	T	F	F	T	F	F	T	F	F	F	F	F

- 更优化的方法（Galil、Vishkin）从问题的描述出发，研究串匹配的基本性质
  - 串的周期性、前缀（设计思路见p.180-181）

并行算法的一般设计方法

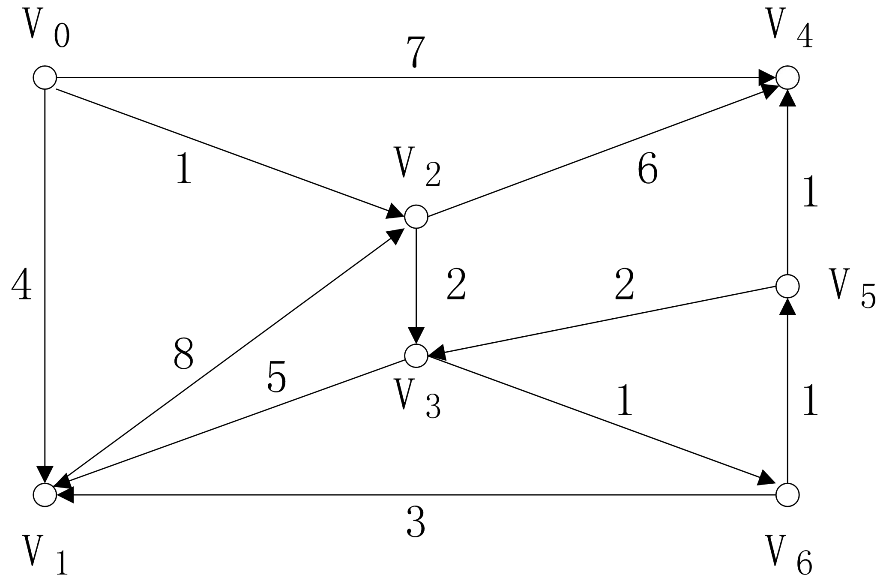
- 串行算法的直接并行化
- 从问题描述开始设计并行算法
- 借用已有算法求解新问题

借用已有算法求解新问题

- 方法描述
  - 找出求解问题和某个已解决问题之间的联系；
  - 改造或利用已知算法应用到求解问题上。
- 评注
  - 这是一项创造性的工作；
  - 范例：使用矩阵乘法算法求解所有点对间最短路径

例子：求所有点对间最短路径

- 问题描述：有向图G=(V,E)，边权矩阵 $W=(w_{ij})_{n\times n}$，求最短路径长度矩阵 $D=(d_{ij})_{n\times n}$，d_{ij} 为 v_i 到 v_j 的最短路径长度



(a)

计算原理

- 假定图中无负权有向回路，记$d^{(k)}_{ij}$为v_i到v_j至多有k-1个中间结点的最短路径长，$D^k=(d^{(k)}_{ij})_{n\times n}$，则
  - $d^{(1)}_{ij}=w_{ij}$ 当 i<>j (如果v_i到v_j之间无边存在记为∞)

    $d^{(1)}_{ij}=0$ 当 i=j

  2. 利用组合最优原理：$d^{(k)}_{ij}=\min_{1\le l\le n}\{d^{(k/2)}_{il}+d^{(k/2)}_{lj}\}$ 式(A)

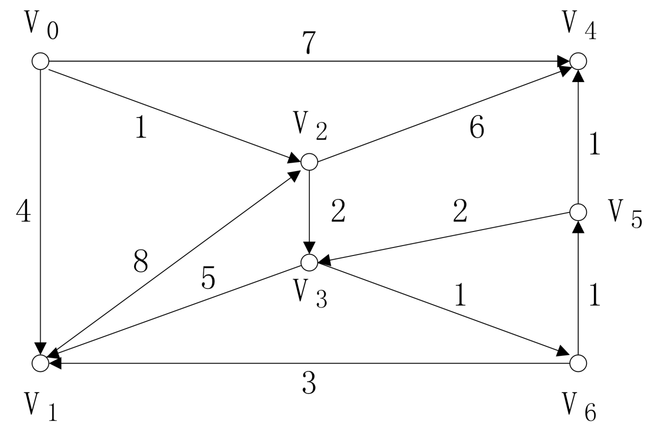
  考虑矩阵乘法D*D，有：$d^{(2)}_{ij}=\sum_{1\le l\le n}(d^{(1)}_{il}*d^{(1)}_{lj})$ 式(B)

  发现式(A)与式(B)的相似之处，视："+" → "×"，"min" → "∑"，则式(A)可看作：

  $$d^{(k)}_{ij}=\sum_{1\le l\le n}\{d^{(k/2)}_{il}\times d^{(k/2)}_{lj}\}$$

  - 应用矩阵乘法：$D^1\rightarrow D^2\rightarrow D^4\rightarrow \ldots\rightarrow D^{2^{\log n}}\ (=D^n)$

图的最短路径示例（1）



(a)

$$
\begin{array}{c}
\begin{array}{c|ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 4 & 1 & \infty & 7 & \infty & \infty \\
1 & \infty & 0 & 8 & \infty & \infty & \infty & \infty \\
2 & \infty & \infty & 0 & 2 & 6 & \infty & \infty \\
3 & \infty & 5 & \infty & 0 & \infty & \infty & 1 \\
4 & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\
5 & \infty & \infty & \infty & 2 & 1 & 0 & \infty \\
6 & \infty & 3 & \infty & \infty & \infty & 1 & 0
\end{array}
\end{array}
$$

D1

$$
\begin{array}{c}
\begin{array}{c|ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 4 & 1 & 3 & 7 & \infty & \infty \\
1 & \infty & 0 & 8 & 10 & 14 & \infty & \infty \\
2 & \infty & 7 & 0 & 2 & 6 & \infty & 3 \\
3 & \infty & 4 & 13 & 0 & \infty & 2 & 1 \\
4 & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\
5 & \infty & 7 & \infty & 2 & 1 & 0 & 3 \\
6 & \infty & 3 & 11 & 3 & 2 & 1 & 0
\end{array}
\end{array}
$$

D2

图的最短路径示例（2）

$$
\begin{array}{c|c}
 & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\
\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} &
\begin{pmatrix}
0 & 4 & 1 & 3 & 7 & 5 & 4 \\
\infty & 0 & 8 & 10 & 14 & 12 & 11 \\
\infty & 6 & 0 & 2 & 5 & 4 & 3 \\
\infty & 4 & 12 & 0 & 3 & 2 & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
0 & 6 & 14 & 2 & 1 & 0 & 3 \\
\infty & 3 & 11 & 3 & 2 & 1 & 0
\end{pmatrix}
\end{array}
$$

D^4

$$
\begin{array}{c|c}
 & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\
\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} &
\begin{pmatrix}
0 & 4 & 1 & 3 & 6 & 5 & 4 \\
\infty & 0 & 8 & 10 & 13 & 12 & 11 \\
\infty & 6 & 0 & 2 & 5 & 4 & 3 \\
\infty & 4 & 12 & 0 & 3 & 2 & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & 6 & 14 & 2 & 1 & 0 & 3 \\
\infty & 3 & 11 & 3 & 2 & 1 & 0
\end{pmatrix}
\end{array}
$$

D^8

并行算法复杂度分析

- **SIMD-CC（立方连接）上的并行算法：算法6.5**

- **时间分析: 每次矩阵乘时间O(log*n*) （调用 P247算法9.6：DNS矩阵乘法），共作$\lceil \log n \rceil$次乘法**

  ==> $t(n)=O(\log^2 n)$, $p(n)=n^3$

课程小结

- 并行计算模型
  - **PRAM模型，BSP模型，logP模型**
  - 三种模型优缺点、比较，及适用范围
- 并行算法的一般设计方法
  - 串行算法的直接并行化
  - 从问题描述开始设计并行算法
  - 借用已有算法求解新问题

推荐网站和读物

- 《并行计算》（第三版）
  - 第5章：并行算法与并行计算模型
  - 第6章：并行算法基本设计策略

- **L.G. Valiant: A Bridging Model for Parallel Computation, Communications of the ACM, Vol. 33, No. 8, pp. 103-111, 1990**
- **LogP: Towards a Realistic Model of Parallel Computation**
- **BSP Worldwide, http://www.bsp-worldwide.org/**
- **Zvi Galil, Optimal parallel algorithms for string matching, Proceedings of the sixteenth annual ACM symposium on Theory of Computing, pp. 240-248, 1984.**

下一讲

- 并行算法设计（2）
  - 《并行计算—结构、算法、编程》（第三版）第7，8章