并 行 计 算

——结构•算法•编程

主讲教师：谢磊

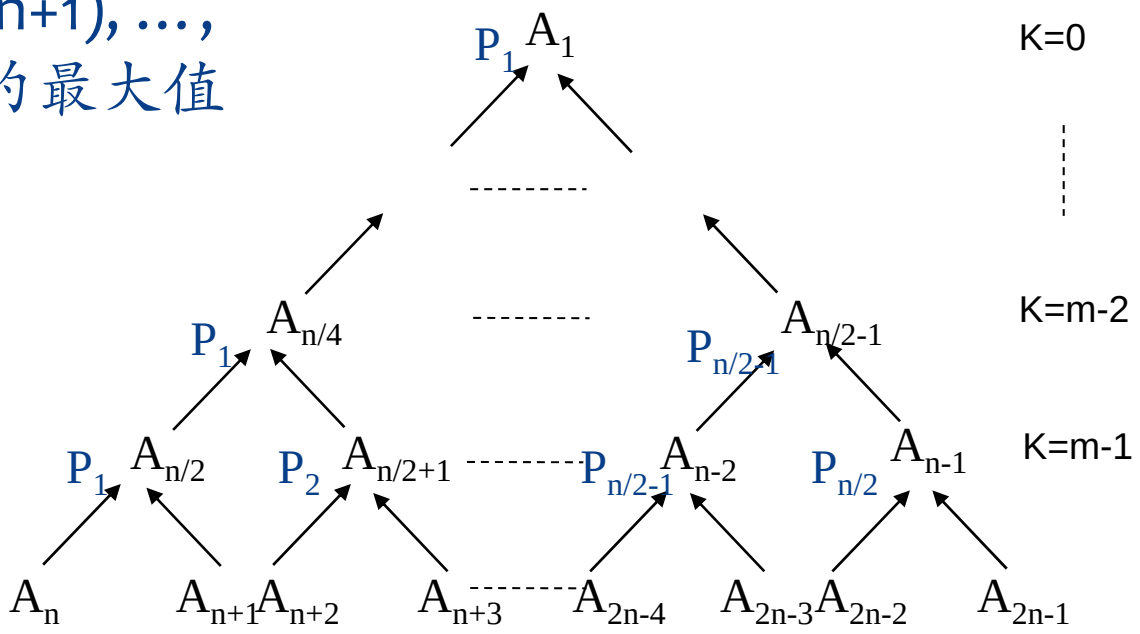
第二篇 并行算法的设计

Case Study

1. 求取最大值算法
2. 计算前缀和算法

求取最大值

* 令n=2^m，A是一个2维的数组，待求最大值的n个数开始存放在A(n), A(n+1), …, A(2n-1)，所求得的最大值置于A(1)中。



求最大值

* 算法4.1：SIMD-TC(SM)上求最大值算法

输入： $n=2^m$ 个数存放在A(n,2n-1)中；

输出： 求得的最大值置于A(1)中。

```
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

算法的时间： $t(n)=m\times O(1)=O(\log n)$；

总比较次数： $O(n)$；

最大的处理器数： $p(n)=n/2$

计算前缀和

* 问题定义

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

  $S_i=x_1*x_2*\cdots*x_i$，$1\leqslant i\leqslant n$　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：SIMD-TC上非递归算法

  令$A[i]=x_i$, i=1~n,

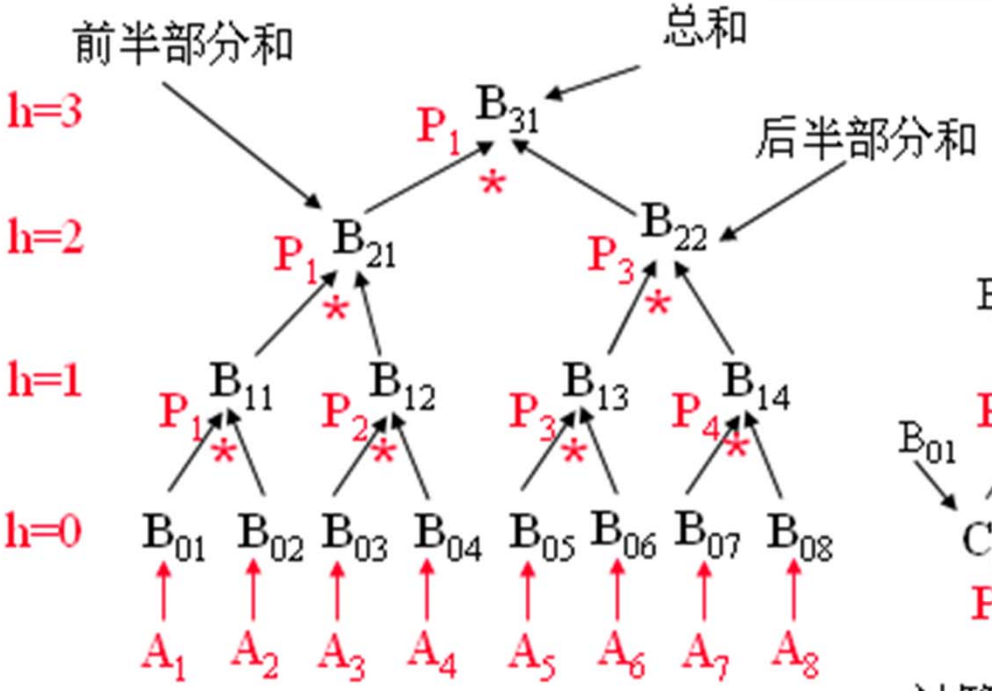
  B[h,j]和C[h,j]为辅助数组(h=0~logn, $j=1\sim n/2^h$)

  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)

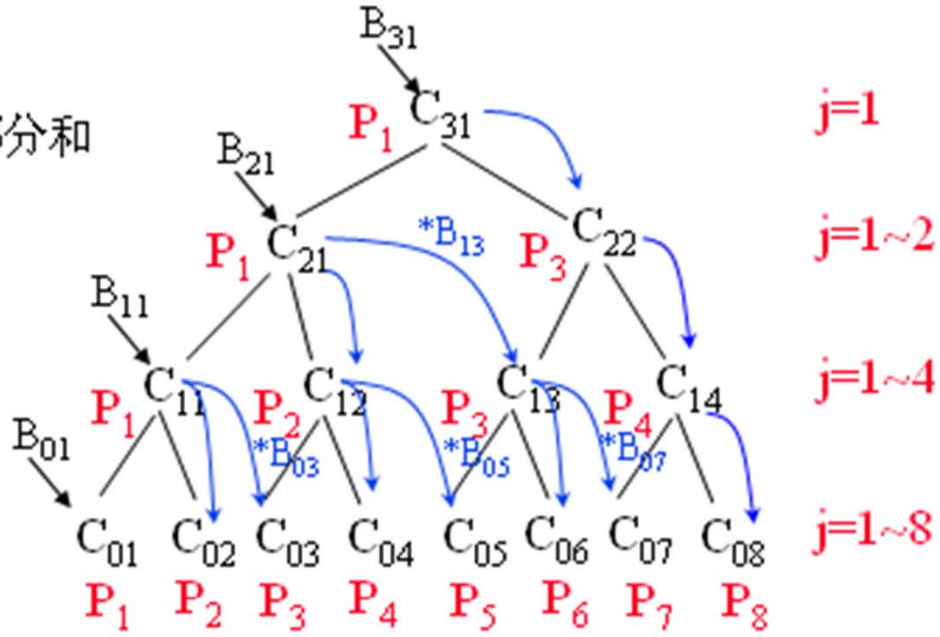
计算前缀和

* 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)

第二篇 并行算法的设计

第四章 并行算法的设计基础

第五章 并行算法的一般设计方法

第六章 并行算法的基本设计技术

第七章 并行算法的一般设计过程

第四章 并行算法的设计基础

4.1 并行算法的基础知识

4.2 并行计算模型

4.1 并行算法的基础知识

4.1.1 并行算法的定义和分类

4.1.2 并行算法的表达

4.1.3 并行算法的复杂性度量

4.1.4 并行算法中的同步和通讯

并行算法的定义和分类

* 并行算法的定义
  * 算法
  * 并行算法：一些可同时执行的诸进程的集合，这些进程互相作用和协调动作从而达到给定问题的求解。
* 并行算法的分类
  * 数值计算和非数值计算
  * 同步算法和异步算法
  * 分布算法
  * 确定算法和随机算法

4.1 并行算法的基础知识

4.1.1 并行算法的定义和分类

4.1.2 并行算法的表达

4.1.3 并行算法的复杂性度量

4.1.4 并行算法中的同步和通讯

并行算法的表达

* 描述语言
  * 可以使用类Algol、类Pascal等；
  * 在描述语言中引入并行语句。
* 并行语句示例
  * Par-do语句

```
for i=1 to n par-do
    ……
end for
```

  * for all语句

```
for all Pi, where 0≤i≤k
  ……
end for
```

4.1 并行算法的基础知识

4.1.1 并行算法的定义和分类

4.1.2 并行算法的表达

4.1.3 并行算法的复杂性度量

4.1.4 并行算法中的同步和通讯

并行算法的复杂性度量

* 串行算法的复杂性度量
  * 最坏情况下的复杂度(Worst-CASE Complexity)
  * 期望复杂度(Expected Complexity)
* 并行算法的几个复杂性度量指标
  * 运行时间t(n):包含计算时间和通讯时间，分别用计算时间步和选路时间步作单位。n为问题实例的输入规模。
  * 处理器数p(n)
  * 并行算法成本c(n): c(n)=t(n)p(n)
    * 如果一个求解问题的并行算法之成本，在数量级上等于最坏情况下串行求解此问题所需的执行步数，则称此并行算法是成本最优（Cost Optimal）的。
  * 总运算量W(n): 并行算法求解问题时所完成的总的操作步数。

并行算法的复杂性度量

* Brent定理

  令W(n)是某并行算法A在运行时间T(n)内所执行的运算量，则A使用p台处理器可在t(n)=O(W(n)/p+T(n))时间内执行完毕。

  * W(n)和c(n)密切相关，c(n)=t(n)*p=O(W(n)+p*T(n))
  * p=O(W(n)/T(n))时，W(n)和c(n)两者是渐进一致的
  * 对于任意的p，c(n)›W(n)。这说明一个算法在运行过程中，不一定都能充分的利用有效的处理器去工作。

4.1 并行算法的基础知识

4.1.1 并行算法的定义和分类

4.1.2 并行算法的表达

4.1.3 并行算法的复杂性度量

4.1.4 并行算法中的同步和通讯

并行算法的同步

* 同步概念
  * 同步是在时间上强使各执行进程在某一点必须互相等待;
  * 可用软件、硬件和固件的办法来实现。
* 同步语句示例
  * 算法4.1 共享存储多处理器上求和算法

    输入：$A=(a_0,\ldots,a_{n-1})$,处理器数p

    输出：$S=\Sigma a_i$

Begin
(1)S=0
(2)for all *Pi* where $0 \leqslant i \leqslant p-1$ do
  (2.1) *L*=0
  (2.2) for *j*=*i* to *n* step *p* do
    $L=L+a_j$
  end for
(2.3) lock(S)
  S=S+*L*
(2.4) unlock(S)
end for
End

并行算法的通讯

* 通讯
  * 共享存储多处理器使用：global read(X,Y)和global write(X,Y)
  * 分布存储多计算机使用：send(X,i)和receive(Y,j)
* 通讯语句示例
  * 算法4.2 分布存储多计算机上矩阵向量乘算法
  * 计算AX=Y,A为n*n的矩阵，X为n*1的矩阵，Y为n*1的矩阵

并行算法的通讯

解决思路

AX

=[A1,A2,A3,…,Ap][X1,X2,X3…,Xp]

=A1*X1+A2*X2+…+Ap*Xp

其中Ai为n*r大小的子矩阵

Xi为r*1大小的子矩阵。

Pi计算Ai(n*r)*Xi(r*1)=Yi(n*1),

在计算y=Y1+Y2+…+Yi,并向右传送此结果；算法结束时，Pi保留乘积AX

输入：处理器数p,

A划分为B=A[1..n,(i-1)r+1..ir],

x划分为w=w[(i-1)r+1;ir]

输出：P_1保存乘积AX

Begin

(1) Compute $z=Bw$

(2) if $i=1$ then $y_i=0$ else receive(y,*left*) endif

(3) $y=y+z$

(4) send(y,*right*)

(5) if $i=1$ then receive(y,*left*)

End

第四章 并行算法的设计基础

4.1 并行算法的基础知识

4.2 并行计算模型

SISD、MIMD、SIMD、MISD

* 1966年，MichealFlynn根据指令和数据流的概念对计算机的体系结构进行了分类，这就是所谓的Flynn分类法。Flynn将计算机划分为四种基本类型，即SISD、MIMD、SIMD、MISD。
* 传统的顺序执行的计算机在同一时刻只能执行一条指令（即只有一个控制流）、处理一个数据（即只有一个数据流），因此被称为单指令流单数据流计算机（Single Instruction Single Data, SISD）。

SISD、MIMD、SIMD、MISD

* 而对于大多数并行计算机而言，多个处理单元都是根据不同的控制流程执行不同的操作，处理不同的数据，因此，它们被称作是多指令流多数据流计算机，即MIMD（Multiple Instruction Multiple Data,MIMD）计算机。
* 曾经在很长一段时间内成为超级并行计算机主流的向量计算机除了标量处理单元之外，最重要的是具有能进行向量计算的硬件单元。在执行向量操作时，一条指令可以同时对多个数据（组成一个向量）进行运算，这就是单指令流多数据流（Single Instruction Multiple Data,SIMD）的概念。因此，我们将向量计算机称为SIMD计算机。

SISD、MIMD、SIMD、MISD

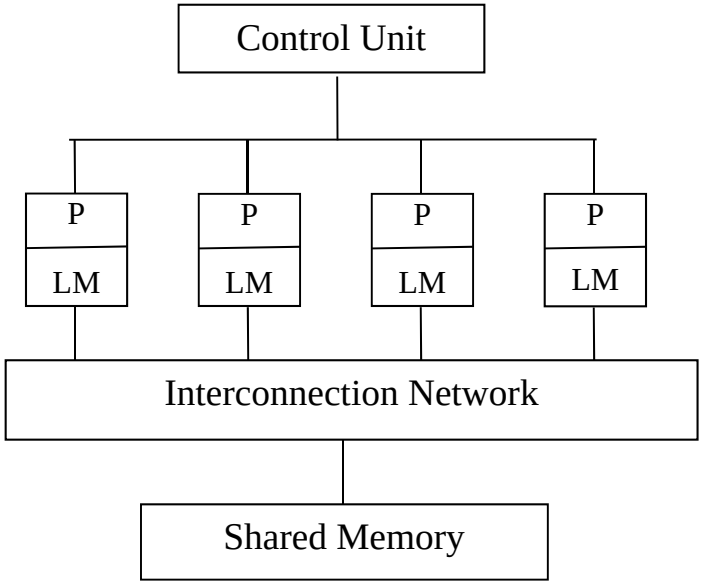
* 第四种类型即所谓的多指令流单数据（Multiple Instruction Single Data, MISD）计算机。在这种计算机中，各个处理单元组成一个线性阵列，分别执行不同的指令流，而同一个数据流则顺次通过这个阵列中的各个处理单元。这种系统结构只适用于某些特定的算法。
* 相对而言，SIMD和MISD模型更适合于专用计算。在商用并行计算机中，MIMD模型最为通用，SIMD次之，而MISD最少用。PII的MMX指令采用的是SISD，高性能服务器与超级计算机大多属于MIMD。
*

4.2 并行计算模型

- 4.2.1 PRAM模型
- 4.2.2 异步APRAM模型
- 4.2.3 BSP模型
- 4.2.4 logP模型

PRAM模型

* 基本概念
  * 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SM的R/W交换数据，隐式同步计算。
* 结构图



PRAM模型

* 分类
  (1) PRAM-CRCW并发读并发写
    * CPRAM-CRCW(Common PRAM-CRCW)：仅允许写入相同数据
    * PPRAM-CRCW(Priority PRAM-CRCW)：仅允许优先级最高的处理器写入
    * APRAM-CRCW(Arbitrary PRAM-CRCW)：允许任意处理器自由写入
  (2) PRAM-CREW并发读互斥写
  (3) PRAM-EREW互斥读互斥写

PRAM模型

* 计算能力比较
  * PRAM-CRCW是最强的计算模型，PRAM-EREW可logp倍模拟PRAM-CREW和PRAM-CRCW

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

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

* 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，忽略了SM的竞争、通讯延迟等因素

4.2 并行计算模型

4.2.1 PRAM模型

4.2.2 异步APRAM模型

4.2.3 BSP模型

4.2.4 logP模型

异步APRAM模型

* 基本概念
  * 又称分相（Phase）PRAM或MIMD-SM。每个处理器有其局部存储器、局部时钟、局部程序；无全局时钟，各处理器异步执行；处理器通过SM进行通讯；处理器间依赖关系，需在并行程序中显式地加入同步路障。
* 指令类型

(1)全局读　　(2)全局写

(3)局部操作　　(4)同步

异步APRAM模型

* 计算过程

由同步障分开的全局相组成

	处理器 1	处理器 2	…	处理器 p
	read x_1	read x_3		read x_n
phase1	read x_2	*		*
	*	write to B		*
	write to A	write to C		write to D
同步障				
	read B	read A		read C
phase2	*	*		*
	write to B	write to D		
同步障				
	*	write to C		write to B
	read D			read A
				write to B
同步障				

异步APRAM模型

* 计算时间
  * 设局部操作为单位时间；全局读/写平均时间为d，d随着处理器数目的增加而增加；同步路障时间为B=B(p)非降函数。满足关系 $2 \le d \le B \le p$；$B(p) \in O(d \log p)$ 或 $O(d \log p / \log d)$ 。
  * 令 t_{ph} 为全局相内各处理器执行时间最长者，则APRAM上的计算时间为 $T = \sum t_{ph} + B \times$同步障次数
* 优缺点
  * 易编程和分析算法的复杂度，但与现实相差较远，其上并行算法非常有限，也不适合MIMD-DM模型。

4.2 并行计算模型

4.2.1 PRAM模型

4.2.2 异步APRAM模型

4.2.3 BSP模型

4.2.4 logP模型

BSP模型

* 基本概念
  * 由Valiant(1990)提出的，“块”同步模型，是一种异步MIMD-DM模型，支持消息传递系统，块内异步并行，块间显式同步。
* 模型参数
  * *p*：处理器数(带有存储器)
  * *l*：同步障时间(Barrier synchronization time)
  * *g*：带宽因子(time steps/packet)=1/bandwidth

BSP模型

* 计算过程

  由若干超级步组成，

  每个超级步计算模式为左图

* 优缺点

  强调了计算和通讯的分离，提供了一个编程环境，易于程序复杂性分析。但需要显式同步机制，限制至多h条消息的传递等。

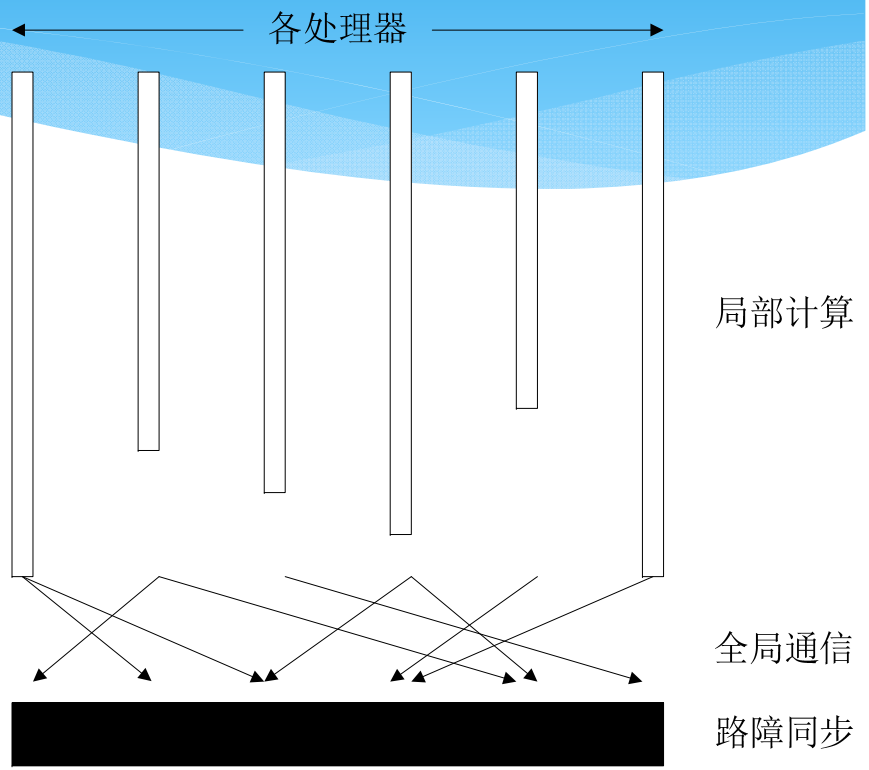


图4.3

4.2 并行计算模型

4.2.1 PRAM模型

4.2.2 异步APRAM模型

4.2.3 BSP模型

4.2.4 logP模型

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*：network latency
  * *o*：communication overhead
  * *g*：gap=1/bandwidth
  * *P*：#processors

注：*L*和g反映了通讯网络的容量

logP模型

* 优缺点

捕捉了MPC的通讯瓶颈，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；但难以进行算法描述、设计和分析。

* **BSP vs. LogP**
  * BSP→LogP：BSP块同步→BSP子集同步→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提供了更方便的程设环境，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

小结

* 并行计算模型综合比较一览表

	PRAM	APRAM	BSP	logP
体系结构	SIMD-SM	MIMD-SM	MIMD-DM	MIMD-DM
计算模式	同步计算	异步计算	异步计算	异步计算
同步方式	自动同步	路障同步	路障同步	隐式同步
模型参数	1 （单位时间步）	d,B d:读/写时间 B:同步时间	p, g, l p:处理器数 g:带宽因子 l:同步间隔	l, o, g, p l:通信延迟 o:额外开销 g:带宽因子 p:处理器数
计算粒度	细粒度/中粒度	中粒度/大粒度	中粒度/大粒度	中粒度/大粒度
通信方式	读/写共享变量	读/写共享变量	发送/接收消息	发送/接收消息
编程地址空间	全局地址空间	单一地址空间	单地址/多地址空间	单地址/多地址空间

小结

* 并行计算模型是设计与分析并行算法的基础。更实际的计算模型成为当今并行算法研究的主要动向之一。
  * PRAM模型过于抽象，不能很好的反映并行算法的实际运行性能。
  * APRAM、BSP、logP模型考虑通信、同步等因素，从而能够较真实的反映并行算法的性能。
  * 近期的研究热点：模型上的算法理论分析转向具体的编程，即从理论研究转向实际应用。