一、定义

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

- 物理世界：连续，数值计算：矩阵，多项式，方程组
- 信息世界：离散，非数值计算：排序、选择、搜索、匹配

分类

- 同步算法：诸进程必须相互等待（电话）
- 异步算法：不必相互等待（短信）
- 数值计算：矩阵，多项式，方程组
- 非数值计算：排序，搜索，匹配
- 分布算法：指由通信链路连接的多个场点（Site）或节点，协同完成问题求解的一类并行算法。
- 确定算法：每一步都能明确地指明下一步应该如何行进
- 随机算法

伪代码

```
* 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
```

二、复杂性度量

串行算法：worst-case + expected（期望）

并行算法：

- 运行时间：计算时间+通讯时间
- 处理器数p
- 并行算法成本c=tp
- 总运算量W：总操作步数

Brent 定理(Brent's Theorem):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。按照成本之定义和 Brent 定理,则有 $c(n)=t(n)p=O(W(n)+pT(n))$,当 $p=O(W(n)/T(n))$时,$W(n)$和 $c(n)$两者是渐近一致的;而对于任意的 p,则 $c(n)>W(n)$。这说明一个算法在运行过程中,不一定都能充分地利用有效的处理器去工作。

三、同步和通信

- 同步语句lock和unlock确保对共享可写数据的互斥访问

访问。假定系统中有 p 个处理器 $P_0,\cdots,P_{p-1}$;输入数组 $A=(a_0,\cdots,a_{n-1})$存放在共享存储器中;全局变量用于存放结果;局部变量 L 包含各处理器计算的子和;lock 和 unlock 语句执行在临界区,加锁是个原子操作;在 for 循环中各进程异步地执行各语句,并结束在"endfor"处。

```
Begin
(1)S=0
(2)for all Pi where 0≤i≤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
```

- 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)

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：P1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
```

四、模型

基础

- SISD单指令流单数据流计算机

> 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

- MIMD多指令流多数据流计算机

> 大多数并行计算机
> 多个处理单元根据不同控制流程执行不同操作，处理不同数据

- SIMD单指令流多数据流计算机

> 在执行向量操作时，一条指令可以同时对多个数据

- MISD多指令流单数据流计算机

> 各个处理单元组成一个线性阵列，分别执行不同指令流
> 同一数据流顺次通过阵列中各个处理单元。
> 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。

PRAM

- 并行随机存取机，共享存储的SIMD，SIMD-SM

* 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-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

异步APRAM

- MIMD-SM

> 分别干活，几天同步一下

2. APRAM 模型中的指令类型

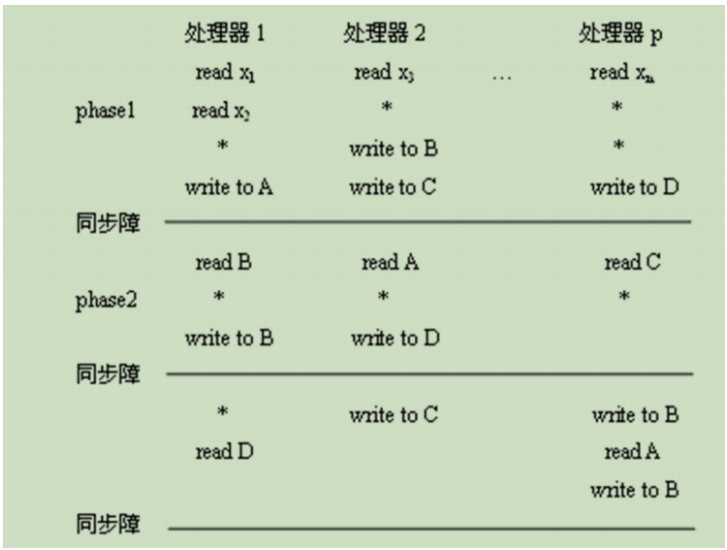
APRAM 模型中的 4 类指令有① **全局读**：将全局存储单元中的内容读入局存单元中；② **局部操作**：对局存中的数执行操作，其结果存入局存中；③ **全局写**：将局存单元中的内容写入全局存储单元中；④ **同步**：同步是计算中的一个逻辑点，在该点各处理器均需等待别的处理器到达后才能继续执行其局部程序。

每个局部程序中的最后一条指令：一条同步障指令Barrie

各处理器均可异步地读取和写入全局存储器SM，但在同一相内不允许两个处理器访问同一单元。

不同的处理器访问存储单元总是由一同步障所分开

指令完成的时间上的差异并不影响整个计算。



* 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。

BSP

- MIMD-DM

> 全球会议，讨论板

* 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

计算系由一系列用全局同步分开的、周期为L的超级步

* 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。

logP

- MIMD-DM

①L（Latency）消息从源到目的地所产生的延迟;

②o（Over-head）处理器发送或接收一条消息所需的额外开销（包含操作系统核心开销和网络软件开销），在此期间内它不能进行其他操作;

③g（Gap）可连续进行消息发送或接收的最小时间间隔;

④p（Processor）表示处理器/存储器模块数。

> g的倒数相应于处理器的通信带宽
> L 和g 反映了通信网络的容量。
> L，o和g 都可以表示成处理器周期（假定一个周期完成一次局部操作，并定义为一个时间单位）的整倍数。

- 在网络容限范围内，点到点的传输一条消息的时间为2o十L。

* 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

* 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。