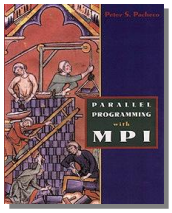
第七章并行计算

一、什么是并行计算

并行:古老的思想!

"...并行计算并不是什么新的思想,只是将它扩展应用于计算机而已". 作者也不认为这种扩展应用会存在什么无法克服的困难. 但也不要期待有效的并行编程方法与技术能够在一夜之间诞生. 期间还需要有许多的工作和实验要做. 毕竟, 今天的编程技术(串行)是若干年来艰苦的探索才取得的. 现在编程工作似乎成了一种令人单调乏味的工作,事实上,并行编程的出现将会使重新恢复编程工作者们的探索精神 ..." (Gill, S. (1958), "Parallel Programming" The Computer Journal, vol. 1, April, pp. 2-10.)



Parallel Programming with MPI
by Peter Pacheco(2000)

并行计算的概念

- 传统上，一般的软件设计都是串行式计算：软件在一台只有一个CPU的电脑上运行；问题被分解成离散的指令序列；指令被一条接一条的执行；在任何时间CPU上最多只有一条指令在运行。
- **并行计算是在串行计算的基础上演变而来，它努力仿真自然世界中的事务状态：一个序列中众多同时发生的、复杂且相关的事件。**

超级计算机

高性能计算机：

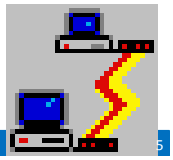
一般指高性能并行计算机系统，大众化的称谓就是超级计算机。

并行处理：

是指同时对多个任务或多条指令、或同时对多个数据项进行处理。

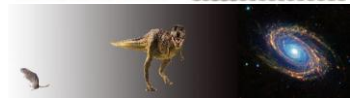
完成此项处理的计算机系统称为并行高性能处理计算机系统。

- 并行计算的基本思想是用多个处理器来协同求解同一问题，*即将被求解的问题分解成若干个部分，各部分均由一个独立的处理器来并行计算。*并行计算系统既可以使专门设计的、含有多个处理器的超级计算机，也可以是以某种方式互连的若干台独立计算机构成的集群。



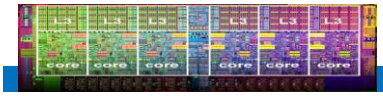
为什么要做并行计算？

节省时间（金钱）



解决大规模问题

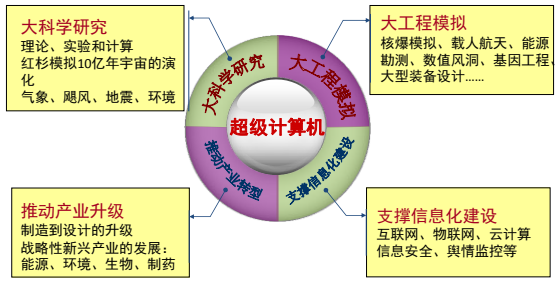
提供高的并发度



串行技术发展受限

- 问题: 科学和工程问题的数值模拟与仿真
  - 计算密集
  - 数据密集
  - 网络密集
  - 三种混合
- 要求:在合理的时限内完成计算任务
  - 秒级　　制造业
  - 分钟级　　短时天气预报(当天)
  - 小时级　　中期天气预报(3~10日)
  - 尽可能快　　长期天气预报(气候)
  - 可计算　　湍流模拟

7



8

- 大工程：高置信度、全三维、全过程数值模拟及大数据处理
  - 载人航天、深海探索、地球模拟
  - 核电站工程
  - 能源勘探、新材料、低温磁性材料模拟
  - 数值风洞、引擎燃烧3D模拟
  - 基因工程、蛋白质工程（凤凰工程）、分子动力学模拟
  - 大型装备设计制造（航母工程）
  - 密码工程.....

9

- 产业升级：
  - 制造到设计的升级：物理模型、计算方法、数值模拟、数字制造……
  - 战略性新兴产业的创新发展：能源、环境、生物、制药……
    - ✓ 天河一号：1060平方公里油气田探勘测数据处理和分析，30天→8天
- 信息化建设：
  - 信息化社会：互联网、物联网、云计算 ……
    - ✓ 人均物均计算和存储能力需求刚刚起步，远远没有满足

10

- “挑战性”问题：
  - 计算能力的挑战：*大计算*
    - ✓ 问题规模大、精度要求高、计算量十分庞大
    - ✓ 能否计算？能否在有效时间内完成计算？
  - 数据处理能力的挑战：*大数据*
    - ✓ 海量数据、强实时输入输出、需要数据挖掘、需要数据融合
    - ✓ 能否存得下？能否在有效时间内处理？能否容易理解和应用？
  - 某些问题还同时存在上述两个方面挑战，变得更具挑战性
- “挑战性”问题牵引超级计算机的发展！
  - 是“矛”与“盾”的关系，是“道”与“魔”的关系，二者对立统一，在相互依存、相互作用中螺旋式上升

11

成功开展并行计算必须具备三个基本条件

- 并行机。并行机至少包含两台或两台以上处理机，这些处理机通过互连网相互连接，相互通信。
- **应用问题必须具有并行度。也就是说，应用可以分解为多个子任务，这些子任务可以并行地执行。将一个应用分解为多个子任务的过程，称为并行算法的设计。**
- 并行编程。在并行机提供的并行编程环境上，具体实现并行算法，编制并行程序，并运行该程序，从而达到并行求解应用问题的目的。

12

超级计算机难在什么地方？

研制难，应用难，基础弱，人才缺！

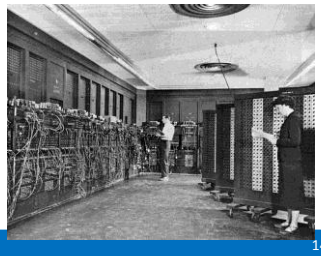
- **研制难：世界上只有几家能干，数不出十个手指头！**
  - 需要先进的使能技术：CPU、互连、控制芯片等核心器件无所不用其极
  - 需要高效的并行计算：串行到并行，通信协同、高效算法、正确性
  - 需要平衡的系统设计：能力、功耗、稳定性、成本、维护
  - 需要实用好用的软件环境：尽可能把复杂人机环境简单化、“傻瓜化”
- **应用难：美欧日领先，仍是难点，我国尤其落后**
  - 各行各业大型并行应用软件的普遍缺失
  - 与应用领域的研究水平密切相关
  - 多学科交叉，并行计算带来的方法学改变，软件复杂、技术门槛高
- **基础弱：我国自然科学落后，产业发展不平衡**
  - 科研和理论基础薄弱，应用领域的模型和算法亟需突破
  - 产业发展不平衡，“制造”亟需向“设计”转型升级
- **人才缺：我国亟需超级计算专业人才、复合型人才**
  - 美国有超过1万人的超级计算高级专业人才，中国？
  - 多学科交叉的复合型人才非常缺乏

二、并行计算机的发展

❑ 始于70 年代

- 1946年第一台计算机 ENIAC
  （ Electronic Numerical Integrator And Computer ）

- 占地170平方
- 重约 30 吨
- 5000 次加法/秒
  或500次乘法/秒
- 15分钟换一个零件
- 主要用于弹道计算
  和氢弹研制



❑ 始于70 年代

- 1972年第一台并行计算机 ILLIAC IV（伊利诺依大学）

- 60 年代末开始建造
- 72 年建成, 74 年运行第一个完整程序, 76年运行第一个应用程序
- 64 个处理器，是当时性能最高的CDC7600机器的 2-6倍
- 公认的1981年前最快
- 1982年退役
- 可扩展性好，
  但可编程性差

❑ 始于70 年代

- 向量机 Cray-1

- 一般将 Cray-1 投入运行的 1976 年称为“ 超级计算元年”
- 编程方便，
  但可扩展性差
- 以 Cray 为代表的向量机称雄超级计算机界十几载

收藏于 Deutsches Museum 德意志博物馆的 Cray-1原型

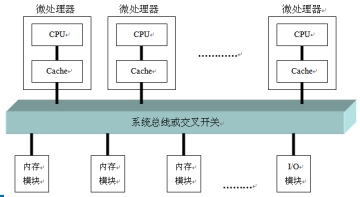
❑ 80 年代百家争鸣

- 早期：以 MIMD 并行计算机的研制为主
  - Denelcor HEP （1982年）第一台商用 MIMD 并行计算机
  - IBM 3090 80 年代普遍为银行所采用
  - Cray X-MP Cray 研究公司第一台 MIMD 并行计算机

西摩 • 克雷 Seymour Cray (1925-1996)，
电子工程学学士，应用数学硕士，
超级计算机之父， Cray研究公司的创始人，
亲手设计了Cray机型的全部硬件与操作系统，
作业系统由他用机器码编写完成。1984年时，
公司占据了超级计算机市场 70%的份额。
1996年Cray研究公司被SGI收购，2000年被
出售给Tera计算机公司，成立Cray公司。

❑ 80 年代百家争鸣

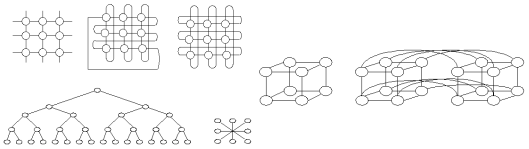
- 中期：共享存储多处理机 Shared-Memory MultiProcessor
  - SMP (Symmetrical Multi-Processing)：在一个计算机上汇集一组处理器，各处理器对称共享内存及计算机的其他资源，由单一操作系统管理，极大提高整个系统的数据处理能力。



- 扩展性较差
- 可靠性较差
- 内存访问瓶颈

❑ 80 年代百家争鸣

- 后期：具有强大计算能力的并行机
  - 通过二维Mesh连接的Meiko（Sun）系统
  - 超立方体连接的 MIMD 并行机：nCUBE-2、iPSC/80
  - 共享存储向量多处理机 Cray Y-MP
  - … …

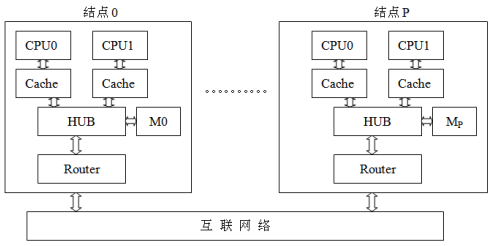


❑ 90 年代：体系结构框架趋于统一 (DSM、MPP、NOW)

- DSM （Distributed Shared Memory）分布式共享存储
  - 以结点为单位，每个结点有一个或多个CPU
  - 专用的高性能互联网络连接（Myrinet, Infiniband, … ）
  - 分布式存储：内存模块局部在每个结点中
  - 单一的操作系统
  - 单一的内存地址空间：所有内存模块都由硬件进行了统一的编址，各个结点既可以直接访问局部内存单元，又可以直接访问其他结点的局部内存单元
  - 可扩展到上百个结点
  - 支持消息传递、共享存储并行程序设计

❑ 90 年代：体系结构框架趋于统一

- DSM （Distributed Shared Memory）分布式共享存储

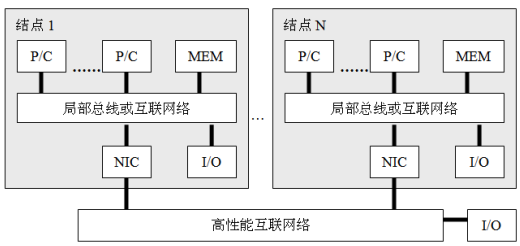


❑ 90 年代：体系结构框架趋于统一

- MPP（Massively Parallel Processing）大规模并行处理结构
  - 每个结点相对独立，有一个或多个微处理器
  - 每个结点均有自己的操作系统
  - 各个结点自己独立的内存，避免内存访问瓶颈
  - 各个结点只能访问自己的内存模块
  - 扩展性较好
  - DM-MPP 分布式存储 MPP：每个结点仅包含一个微处理器
  - SMP-MPP：每个结点是一台 SMP 并行机
  - DSM-MPP：每个结点是一台 DSM 并行机

❑ 90 年代：体系结构框架趋于统一

- MPP体系结构示意图

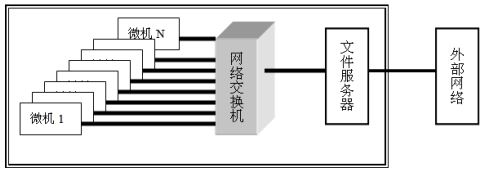


❑ 90 年代：体系结构框架趋于统一

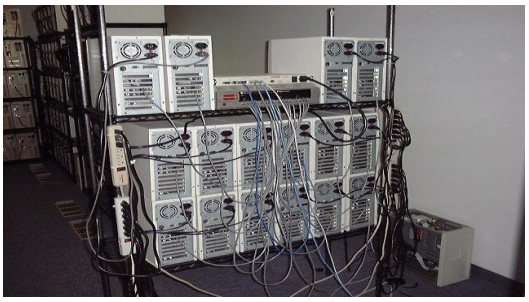
- NOW（Network of Workstations）工作站机群
  - 每个结点都是一个完整的工作站，有独立的硬盘与UNIX系统
  - 结点间通过低成本的网络（如千兆以太网）连接
  - 每个结点安装消息传递并行程序设计软件，实现通信、负载平衡等
  - 投资风险小、结构灵活、可扩展性强、通用性好、异构能力强，被大量中小型计算用户和科研院校所采用
  - 也称为 COW（Cluster of Workstations）
  - NOW（COW）与 MPP 之间的界线越来越模糊

❑ 90 年代：体系结构框架趋于统一

- NOW的典型代表：Beowulf cluster 微机机群
  - 性能价格比极高



第一台 Beowulf 机群



❑ 2000 年至今：前所未有大踏步发展

- Cluster 机群
  - 每个结点含多个商用处理器，结点内部共享存储
  - 采用商用机群交换机通过前端总线连接结点，结点分布存储
  - 各个结点采用 Linux 操作系统、GNU编译系统和作业管理系统
- Constellation 星群
  - 每个结点是一台子并行机
  - 采用商用机群交换机通过前端总线连接结点，结点分布存储
  - 各个结点运行专用的结点操作系统、编译系统和作业管理系统
- MPP
  - 专用高性能网络，大多为政府直接支持

国际超算发展现状

美国全面领先，欧洲应用水平高，日本有实力！

➢ **TOP2：美国橡树岭国家实验室，**

Cray泰坦（XK7），

17.59P（27P），

8.2MW，

能效比2.15GFlops/W

美国全面领先，欧洲应用水平高，日本有实力！

➢ **TOP3：美国劳伦斯利弗莫国家实验室，**

IBM红杉（蓝色基因），

17.2P（20P），

7.89MW，

能效比2.17GFlops/W

美国全面领先，欧洲应用水平高，日本有实力！

➢ **TOP4：日本先进计算科学研究所，**

Fujitsu“京”（10^{16}），

10P（11P），

12.66MW，

能效比0.8GFlops/W

美国全面领先，欧洲应用水平高，日本有实力！

- TOP5：美国阿岗国家实验室，
  IBM米拉（蓝色基因），
  8.16P（10P），
  3.95MW，
  能效比2.17GFlops/W

31

美国全面领先，欧洲应用水平高，日本有实力！

德国尤利西欧洲超算，
IBM JUQueen（蓝色基因），
4P（5P），
1.97MW，
能效比2.03GFlops/W

32

德国莱布尼兹超级计算中心，
IBM SuperMUC，
2.897P（3.185P），
3.4MW，
能效比0.85GFlops/W

33

- 美国德克萨斯先进计算中心，
  Dell Stampede，2.66P（3.96P）
- 中国国家超级计算天津中心，
  天河一号，2.566P（4.7P），
  4MW，能效比0.64GFlops/W

34

美国全面领先，欧洲应用水平高，日本有实力！

- 意大利西利卡超级计算中心，
  IBM费米（蓝色基因），
  1.725P（2.97P），
  0.822MW，
  能效比2.1GFlops/W

35

三、我国高性能计算机的发展

- 第一台计算机

- 1958年第一台国产计算机
  —— 103型计算机
- 运行速度每秒 1500次

36

➢ 超级计算机

- 银河
- 神威

军事部门

- 曙光（曙光信息产业有限公司）
- 深腾（联想集团）

- 1983年12月，我国第一台每秒钟运算达1亿次以上的计算机“银河 I”问世
- 1992年 “银河-II”问世，每秒运算达10亿次
- 1997年成功研制百亿次并行机 “银河-III”，由130多个处理结点组成

- 1999年9月，由国家并行计算机工程技术研究中心牵头研制成功的 “神威” 计算机系统投入运行。
- 2000年，“神威I”面向社会开放使用。
- “神威I” 的峰值速度为每秒3840亿次浮点运算

神威 I

- 1993年10月研制成功 “曙光一号” SMP多处理机
- 2000年推出每秒 3000 亿次的曙光3000超级服务器
- 2004年6月，推出 11万亿次的曙光4000A超级计算机，落户上海超算中心，进入全球前十名，从而使中国成为继美国和日本之后，第三个能研制10万亿次高性能计算机的国家
- 2008年6月，曙光5000A发布，实际运算速度超过每秒160万亿次，排名世界第十

曙光5000A

- 2002年，联想发布深腾1800计算机，排名全球第43位，成为首家正式进入排行榜前100的中国企业
- 2003年，深腾6800计算机发布，列全世界TOP500第14位，其78.5%的整机效率列世界通用高端计算机第一名
- 2008年12月，联想发布百万亿次超级计算机深腾7000

深腾7000

- **百万亿次以上的部门和行业超级计算中心：**
  - **中国工程物理研究院北京九所：千万亿次银河**
  - **中国空气动力学研究中心：二百万亿次银河**
  - **航天科技集团一院某计算中心：百万亿次银河**
  - **航天科工集团三院某计算中心：百万亿次神威**
  - **总参某部总站计算中心：百万亿次银河**
  - **总参某部北方计算中心：千万亿次神州**
  - **中科院网络与计算中心：百万亿次联想深腾**
  - **清华大学计算中心：百万亿次浪潮**
  - **东方石油公司：百万亿次多品牌复合计算能力**
  - **中国气象局国家气象中心：千万亿次超级升级中**
  - **华大基因：多品牌复合计算能力**
  - **电信行业，互联网应用行业，……**

我国已建成的部分超级计算中心

天津超算	深圳超算	长沙超算	济南超算	上海超算
4700万亿次/2010年6月	1271万亿次/2011年11月	1372万亿次/2011年10月	1070万亿次/2011年	230万亿次/2009年
国防科技大学	曙光公司	国防科技大学	江南计算所	曙光公司
石油勘探数据处理、生物医药研究、航空航天装备研制、资源勘测和卫星遥感数据处理、海洋环境数值模拟、新材料开发和设计	面向地球与大气科学、生物学与药物设计等领域提供高性能计算应用，同时突出云计算技术应用	支撑数字湖南建设、气象公共服务、高端机械装备设计等应用	在海洋研究、高效生态农业、地质勘探、新药创制、生物信息等领域均有应用。	气象预报、药物设计、生命科学、汽车、新材料、土木工程、物理、化学、航空、航天和船舶等

广州超算计算中心

- 国家十二五863重大科技项目
- 广州科技一号工程，总投资约25亿元
- 建设峰值计算性能10亿亿次以上的“天河二号”超级计算机系统
- 计算能力是美国“泰坦”系统的4倍，天津超算23倍

四、并行算法分类

- 按运算对象：数值并行算法、非数值并行算法
- 按并行进程执行顺序：

  同步并行算法、异步并行算法、独立并行算法
- 按计算任务：

  细粒度并行算法（基于向量和循环级并行）

  中粒度并行算法（基于较大的循环级并行）

  大粒度并行算法（基于子任务级并行）

五、并行算法的设计基础

5.1 并行算法的基础知识

5.2 并行计算模型

5.1 并行算法的基础知识

5.1.1 并行算法的定义和分类

5.1.2 并行算法的表达

5.1.3 并行算法的复杂性度量

5.1.4 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

5.1 并行算法的基础知识

5.1.1 并行算法的定义和分类

5.1.2 并行算法的表达

5.1.3 并行算法的复杂性度量

5.1.4 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
```


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)
  - 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：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≤i≤p-1 do
    (2.1) L=0
    (2.2) for j=i to n step p do
        L=L+aj
    end for
  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：分布存储多计算机上矩阵向量乘算法

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 yi=0 else receive(y,left) endif
  (3) y=y+z
  (4) send(y,right)
  (5) if i=1 then receive(y,left)
End
```

六、并行算法的设计基础

6.1 并行算法的基础知识

6.2 并行计算模型

6.2 并行计算模型

6.2.1 PRAM模型

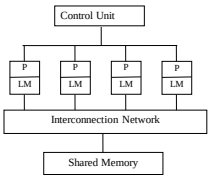
6.2.2 异步APRAM模型

6.2.3 BSP模型

6.2.4 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

- 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

PRAM模型

- 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

6.2 并行计算模型

6.2.1 PRAM模型

6.2.2 异步APRAM模型

6.2.3 BSP模型

6.2.4 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

61

异步APRAM模型

- 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。

62

6.2 并行计算模型
6.2.1 PRAM模型
6.2.2 异步APRAM模型
6.2.3 BSP模型
6.2.4 logP模型

63

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

64

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。

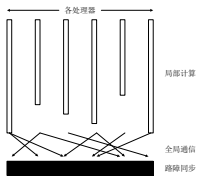


图4. 3

65

6.2 并行计算模型
6.2.1 PRAM模型
6.2.2 异步APRAM模型
6.2.3 BSP模型
6.2.4 logP模型

66

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

七、并行算法的一般设计方法

7.1 串行算法的直接并行化

7.2 从问题描述开始设计并行算法

7.3 借用已有算法求解新问题

7.1串行算法的直接并行化

7.1.1 设计方法描述

7.1.2 快排序算法的并行化

设计方法的描述

- 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。

7.1串行算法的直接并行化

7.1.1 设计方法描述

7.1.2 快排序算法的并行化

快排序算法的并行化

- 算法： PRAM-CRCW上的快排序二叉树构造算法

输入：序列($A_1,\ldots,A_n$)和n个处理器
输出：供排序用的一棵二叉排序树
Begin
(1)for each processor i do
(1.1)root=i
(1.2)f_i=root
(1.3)LC_i=RC_i=n+1
end for
(2)repeat for each processor i<>root do
if (A_i<A_{f_i})∨(A_i=A_{f_i}∧i<f_i) then
(2.1)LC_{f_i}=i
(2.2)if i=LC_{f_i} then exit else f_i=LC_{f_i} endif
else
(2.3)RC_{f_i}=i
(2.4)if i=RC_{f_i} then exit else f_i=RC_{f_i} endif
endif
end repeat
End

七、并行算法的一般设计方法

7.1 串行算法的直接并行化
7.2 从问题描述开始设计并行算法
7.3 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；
  - 利用串的周期性的PRAM-CRCW算法是一个很好的范例；

七、并行算法的一般设计方法

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

7.3 借用已有算法求解新问题

7.3.1 设计方法描述
7.3.2 利用矩阵乘法求所有点对间最短路径

设计方法的描述

- 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；
  - 使用矩阵乘法算法求解所有点对间最短路径是一个很好的范例。

7.3 借用已有算法求解新问题
7.3.1 设计方法描述
7.3.2 利用矩阵乘法求所有点对间最短路径

利用矩阵乘法求所有点对间最短路径

- 计算原理

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。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

(1) $d^{(1)}_{ij}=w_{ij}$　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) 无负权回路 → $d_{ij}=d^{(n-1)}_{ij}$

(3) 利用最优性原理：$d^{(k)}_{ij}=\min_{1\le l\le n}\{d^{(k/2)}_{il}+d^{(k/2)}_{lj}\}$
视：”+” → “×”，“min” → “∑”，则上式变为
$d^{(k)}_{ij}=\sum_{1\le l\le n}\{d^{(k/2)}_{il}\times d^{(k/2)}_{lj}\}$

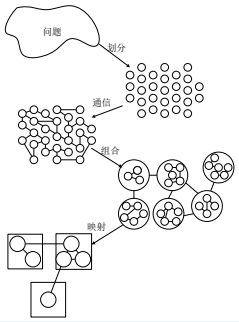
(4) 应用矩阵乘法：$D^1\rightarrow D^2\rightarrow D^4\rightarrow \ldots\rightarrow D^{2^{\log n}}$ $(=D^n)$

八、并行算法的一般设计过程
8.1 PCAM设计方法学
8.2 划分
8.3 通讯
8.4 组合
8.5 映射
8.6 小结

PCAM设计方法学

- 设计并行算法的四个阶段
  - 划分(Partitioning)
  - 通讯(Communication)
  - 组合(Agglomeration)
  - 映射(Mapping)
- 划分：分解成小的任务，开拓并发性；
- 通讯：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



八、并行算法的一般设计过程
8.1 PCAM设计方法学
8.2 划分
8.3 通讯
8.4 组合
8.5 映射
8.6 小结

8.2 划分
- 8.2.1 方法描述
- 8.2.2 域分解
- 8.2.3 功能分解
- 8.2.4 划分判据

划分方法描述

- 充分开拓算法的并发性和可扩放性；
- 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；
- 能分为两类划分：
  - 域分解(domain decomposition)
  - 功能分解(functional decomposition)

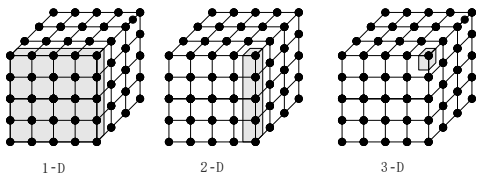
8.2 划分
- 8.2.1 方法描述
- 8.2.2 域分解
- 8.2.3 功能分解
- 8.2.4 划分判据

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讯；

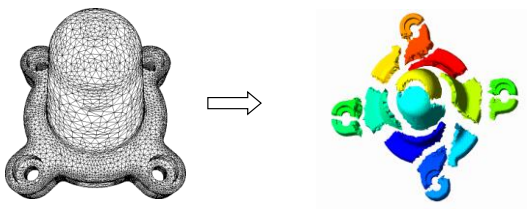
域分解

- 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：



域分解

- 不规则区域的分解示例：



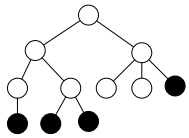
8.2 划分
- 8.2.1 方法描述
- 8.2.2 域分解
- 8.2.3 功能分解
- 8.2.4 划分判据

功能分解

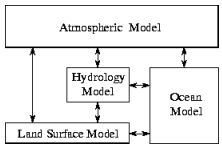
- 划分的对象是计算，将计算划分为不同的任务，其出发点不同于域分解；
- 划分后，研究不同任务所需的数据。如果这些数据不相交的，则划分是成功的；如果数据有相当的重叠，意味着要重新进行域分解和功能分解；
- 功能分解是一种更深层次的分解。

功能分解

- 示例1：搜索树



- 示例2：气候模型



8.2 划分
- 8.2.1 方法描述
- 8.2.2 域分解
- 8.2.3 功能分解
- 8.2.4 划分判据

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？

八、并行算法的一般设计过程

8.1 PCAM设计方法学
8.2 划分
8.3 通讯
8.4 组合
8.5 映射
8.6 小结

8.3 通讯
- 8.3.1 方法描述
- 8.3.2 四种通讯模式
- 8.3.3 通讯判据

通讯方法描述

- 通讯是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；从而产生了通讯；
- 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，通讯则限制了这种并发性；

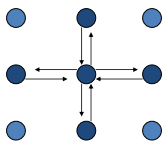
8.3 通讯
- 8.3.1 方法描述
- 8.3.2 四种通讯模式
- 8.3.3 通讯判据

四种通讯模式

- 局部/全局通讯
- 结构化/非结构化通讯
- 静态/动态通讯
- 同步/异步通讯

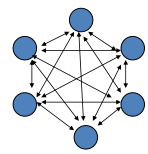
局部通讯

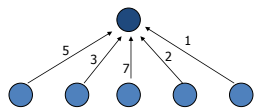
- 通讯限制在一个邻域内



全局通讯

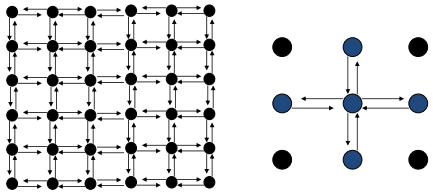
- 通讯非局部的
- 例如：
  - All to All
  - Master-Worker





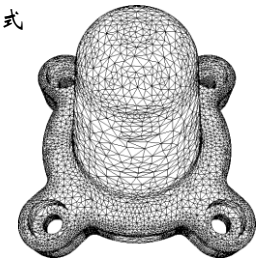
结构化通讯

- 每个任务的通讯模式是相同的；
- 下面是否存在一个相同通讯模式？



非结构化通讯

- 没有一个统一的通讯模式
- 例如：无结构化网格



8.3 通讯
- 8.3.1 方法描述
- 8.3.2 四种通讯模式
- 8.3.3 通讯判据

通讯判据

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

八、并行算法的一般设计过程
- 8.1 PCAM设计方法学
- 8.2 划分
- 8.3 通讯
- 8.4 组合
- 8.5 映射
- 8.6 小结

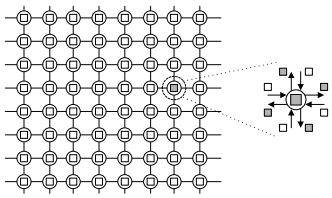
8.4 组合
- 8.4.1 方法描述
- 8.4.2 表面-容积效应
- 8.4.3 重复计算
- 8.4.4 组合判据

方法描述

- 组合是由抽象到具体的过程，是将组合的任务能在一类并行机上有效的执行；
- 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，可以减少通讯成本；
- 保持映射和扩展的灵活性，降低软件工程成本；

表面-容积效应

- 通讯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讯量；

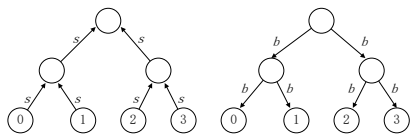


重复计算

- 重复计算减少通讯量，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；

重复计算

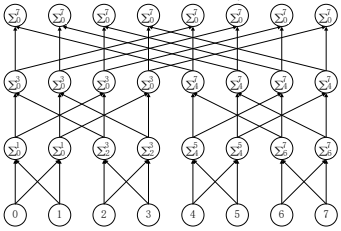
- 示例：二叉树上N个处理器求N个数的全和，要求每个处理器均保持全和。



二叉树上求和，共需2logN步

重复计算

- 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

8.4 组合
- 8.4.1 方法描述
- 8.4.2 表面-容积效应
- 8.4.3 重复计算
- 8.4.4 组合判据

组合判据

- 增加粒度是否减少了通讯成本？
- 重复计算是否已权衡了其得益？
- 是否保持了灵活性和可扩放性？
- 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讯？
- 有没有减少并行执行的机会？

八、并行算法的一般设计过程
- 8.1 PCAM设计方法学
- 8.2 划分
- 8.3 通讯
- 8.4 组合
- 8.5 映射
- 8.6 小结

8.5 映射
- 8.5.1 方法描述
- 8.5.2 负载平衡算法
- 8.5.3 任务调度算法
- 8.5.4 映射判据

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
  - 任务之间存在高通讯的 → 同一处理器
- 映射实际是一种权衡，属于NP完全问题；

8.5 映射
- 8.5.1 方法描述
- 8.5.2 负载平衡算法
- 8.5.3 任务调度算法
- 8.5.4 映射判据

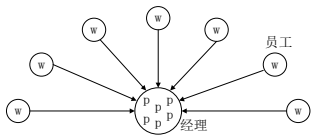
负载平衡算法

- 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

8.5 映射
- 8.5.1 方法描述
- 8.5.2 负载平衡算法
- 8.5.3 任务调度算法
- 8.5.4 映射判据

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

8.5 映射
- 8.5.1 方法描述
- 8.5.2 负载平衡算法
- 8.5.3 任务调度算法
- 8.5.4 映射判据

映射判据

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

小结

- 划分
  - 域分解和功能分解
- 通讯
  - 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