学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.5 多处理机的机间互连形式

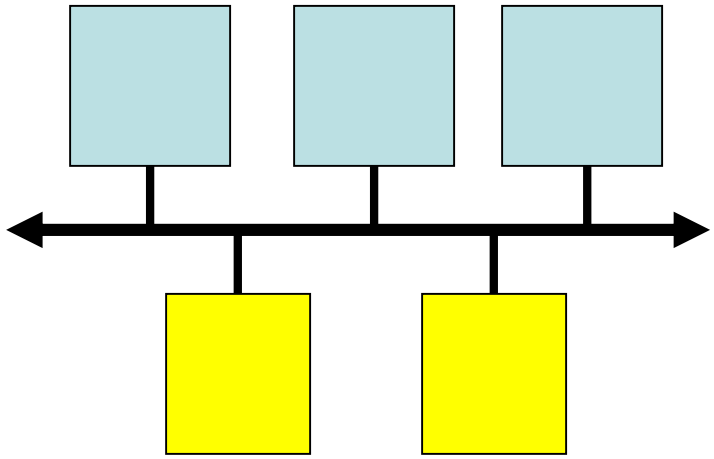
- 互连网络是并行计算机的核心。
- 多处理机的机间互连要求：
  - 高速率
  - 低成本
  - 能实现各种复杂、无规则的互连而不发生冲突
  - 具有良好的可扩展性

7.5 多处理机的机间互连形式

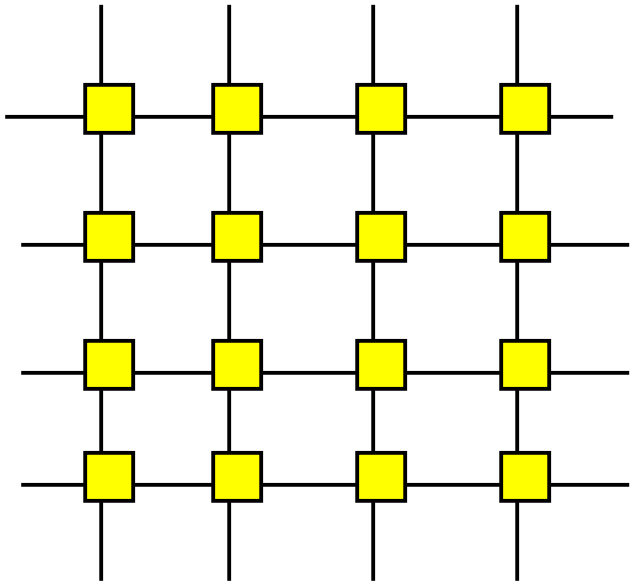
- 主流选择：
  - 总线形 (Bus)
  - 环形 (Ring)
  - 交叉开关 (Crossbar)
  - 带缓存的交叉开关 (Buffered crossbar)
  - 多级交叉开关
  - 多端口存储器
  - 网状 (Mesh)
  - 应用特定的 (Application-specific)

Bus network

- **Advantages:**
  - **Well-understood.**
  - **Easy to program.**
  - **Many standards.**
- **Disadvantages:**
  - **Contention.**
  - **Significant capacitive load.**



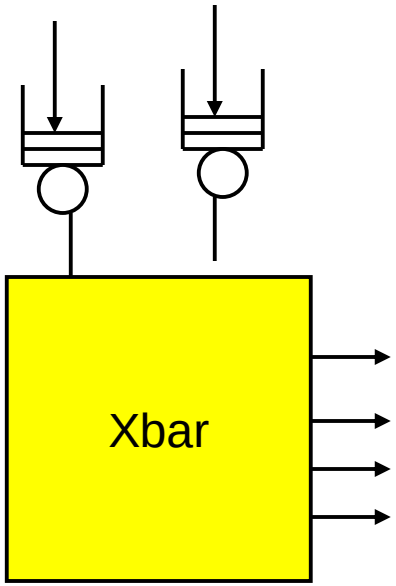
Crossbar



- **Advantages:**
  - **No contention.**
  - **Simple design.**
- **Disadvantages:**
  - **Not feasible for large numbers of ports.**

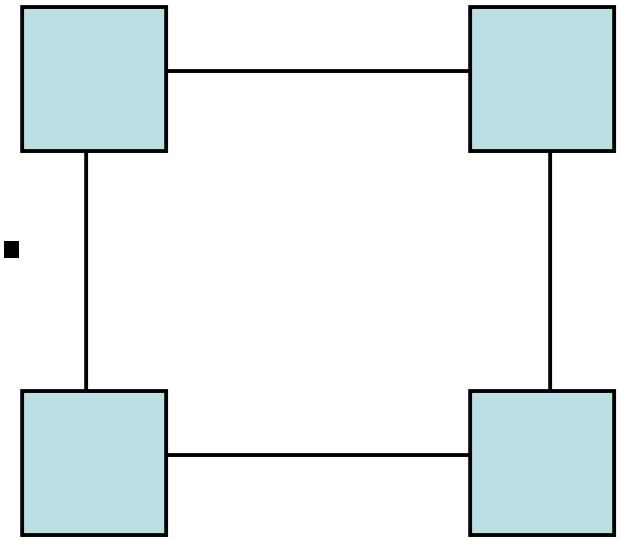
Buffered crossbar

- **Advantages:**
  - **Smaller than crossbar.**
  - **Can achieve high utilization.**
- **Disadvantages:**
  - **Requires scheduling.**



Mesh

- **Advantages:**
  - **Well-understood.**
  - **Regular architecture.**
- **Disadvantages:**
  - **Poor utilization.**



学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.6 程序并行性

- 并行处理面临着两个重要的挑战：
  - 程序中有限的并行性
  - 相对较高的通信开销

$$\text{系统加速比} = \frac{1}{(1-\text{可加速部分比例}) + \frac{\text{可加速部分比例}}{\text{理论加速比}}}$$

并行处理面临的挑战

1. 程序中有限的并行性

- 使机器要达到好的加速比十分困难

例：假设有100个处理器，希望获得80倍的加速比。问原始程序中串行部分的百分比应达到多少？

a. 10%
b. 5%
c. 1%
d. <1%

Amdahl's Law Answers

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{\text{Speedup}_{\text{parallel}}}}$$

$$80 = \frac{1}{\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times \left(\left(1 - \text{Fraction}_{\text{parallel}}\right) + \frac{\text{Fraction}_{\text{parallel}}}{100}\right) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

并行处理面临的挑战

2. 相对较高的通信开销

- 在现有的机器中，处理器之间的数据通信大约需要50～10000个时钟周期。

例： 一台32个处理器的计算机，对远程存储器访问时间为2000ns。除了通信以外，假设计算中的访问均命中局部存储器。当发出一个远程请求时，本处理器挂起。处理器时钟时间为10ns，如果指令基本的CPI为1.0(设所有访存均命中Cache)，求在没有远程访问的状态下与有0.5%的指令需要远程访问的状态下，前者比后者快多少?

并行处理面临的挑战

解：有0.5%远程访问的机器的实际CPI为

CPI＝基本CPI＋远程访问率×远程访问开销

＝1.0 ＋ 0.5%×远程访问开销

远程访问开销＝远程访问时间/时钟时间

＝2000ns/10ns＝200个时钟

∴ CPI＝1.0＋0.5%×200＝2.0

它为只有局部访问的机器的2.0 / 1.0＝2倍，因此在没有远程访问的状态下的机器速度是有0.5%远程访问的机器速度的2倍。

并行处理面临的挑战

解决问题的方法：

- 并行性不足： 采用并行性更好的算法
- 远程访问延迟的降低：靠体系结构支持和编程技术
- 例如：减少远程访问频率的措施：
  - 缓存共享数据 (HW)
  - 合理分布数据，尽可能多地访问本地数据 (SW)
- 目前：使用硬件方法，通过使用Cache减少延迟

并行性开发途径：语言、编译、算法、OS等方面

7.6.1 并行算法

- 算法：
  - 求解问题的方法和步骤。
- 并行算法
  - 用多台处理机联合求解问题的方法和步骤。
- 并行算法与串行算法最大的不同：
  - 并行算法不仅要考虑问题本身，而且还要考虑所使用的并行模型，网络连接等。

并行算法的发展

- 基于向量运算的并行算法设计阶段
- 基于多向量处理机的并行算法设计阶段
- SIMD类并行机上的算法设计阶段
- MIMD类并行机上的并行算法设计阶段
- 现代并行算法设计——以MIMD为主，要求可扩展性、可移植性

并行算法分类

- 根据运算的基本对象的不同分为：
  - 数值并行算法（数值计算）
  - 非数值并行算法（符号计算）
- 根据进程之间的依赖关系分为：
  - 同步并行算法（步调一致）
  - 异步并行算法（步调、进展互不相同）
  - 纯并行算法（各部分之间没有关系）。

并行算法分类

- 根据并行计算任务的大小分为：
  - 粗粒度并行
  - 细粒度并行
  - 中粒度并行

并行的粒度越小，就有可能开发更多的并行性，提高并行度，但是通信次数和通信量就增加很多。

- 并行算法分为：
  - 多机并行
  - 多线程并行

并行算法设计

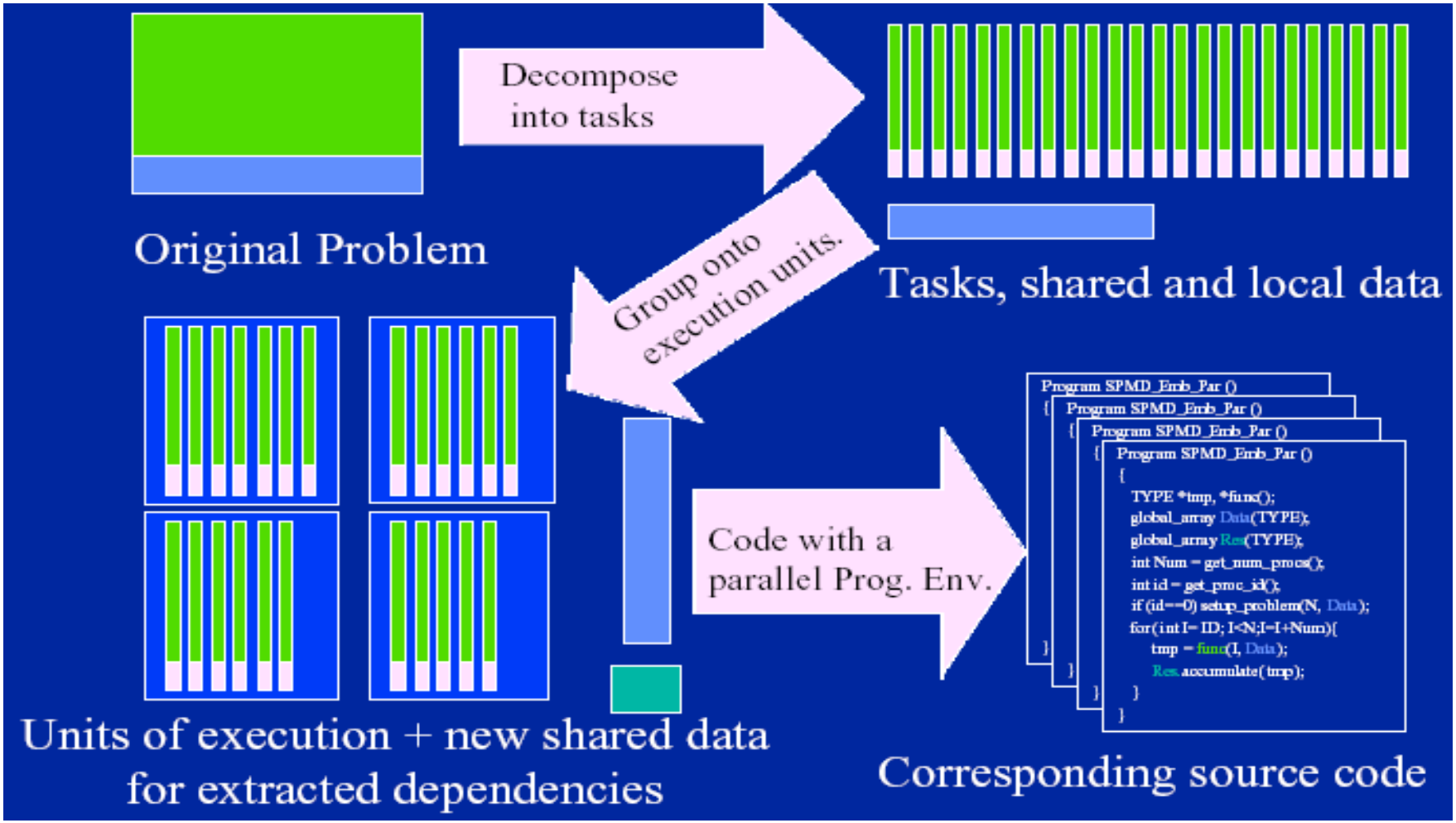
- 并行算法是提高计算机并行性能的关键
- 并行算法设计：
  - 以MIMD 为主
  - 可扩展、可移植
  - 中/大粒度任务级并行
  - 每个进程发挥单机性能 （数据结构、程序设计、通信方式）

并行算法设计

- **并行算法依赖一个简单事实：**独立的计算可同时执行。
- 所谓独立计算是指其每个结果元只出现一次的计算。
- **如何实现并行计算？**

如何实现并行计算?

分而治之!



并行计算基本设计技术(1/3)

- 划分法(Partitioning)
  - 首先，将原问题分成p个独立的近乎大小相等的子问题；其次，用p台处理器并行求解诸子问题。
  - 划分的难点在于要留心分解子问题，使得子问题的解很容易被组合成原问题的解。
  - 例如(m, n)-selection网络。
- 分治法(Divide-and-Conquer)
  - 将原问题规模从大到小逐渐分解成一些特性相同的子问题；直到子问题很容易求解为止。
  - 分治很自然地导致递归过程，其注意力集中在子问题地合并上。
  - 例如FFT的计算。

并行计算基本设计技术 (2/3)

- **平衡树法(Balanced Tree)**
  - 将输入元素作为叶节点构筑一棵平衡二叉树；然后自叶向根往返遍历。
  - 此法的优点是在树中能快速存取所需的信息。
  - 例如数据播送、求最大/最小值以及求和/前缀计算等。
- **倍增法(Doubling)/指针跳跃法(Pointer Jumping)**
  - 使用递归计算，将需要处理的数据间的距离逐步加倍，经k步后就可以完成距离为2^k的所有数据的计算。
  - 此法特别适合于处理以链表或有根树之类为数据结构的问题。
  - 例如表序问题的计算和求森林根等。

并行计算基本设计技术 (3/3)

- **流水线法(Piplining)**
  - 将原任务t分成一系列子任务t_1，t_2，…，t_m，使得一旦t_i完成，后继的子任务就立即开始，并以同样的速率计算之。
  - 例如systolic计算。
- **破对称法(Symmetry Breaking)**
  - 打破某些问题的对称性，使原问题可并行计算。
  - 例如有向环图的顶点着色。

并行算法设计

- 建立并行算法的一种**普遍原则**：
  - 反复将每一计算分裂成具有同等复杂性的两个独立部份，称为**递推倍增法**。
  - 将各部分之间的关联用结点组成的树来描述。**提高并行性就是用交换律、结合律、分配律对树进行变换。**
  - 增大树中每一层的结点数（增大各处理机可并行运行的过程数），降低树的高度（降低多处理机运算的级数）。

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
- 借用已有算法解新问题

并行算法评价

- 性能的参数：
  - P：可以并行处理的处理机数；
  - T_P：P台处理机运算的级数，也就是树高；
  - S_P：加速比，单处理机顺序运算的级数T_1与P台处理机并行运算的级数T_P之比；
  - E_P：效率（P台处理机的设备利用率），$E_P=S_P/P$。

在多处理机上，好的并行算法应当是以提高系统的速度性能为前提，再在此基础上尽可能地提高系统的效率。

7.6.2 程序段间的相关性分析

- 程序段中各类数据的相关是限制程序并行的重要因素。
- 多处理机上的相关 (1/2)：
  - 数据相关：
    - “先写后读”，可以顺序串行，但不能并行
  - 数据反相关
    - “先读后写”，可以顺序串行，不能交换串行，在特殊情况下可以并行。

7.6.2 程序段间的相关性分析

- 多处理机上的相关 (2/2)：
  - 数据输出相关
    - “写–写”，可以顺序串行，不能交换串行，在特殊情况下可以并行。
  - 控制相关
    - 条件语句
  - 相互交换
    - 同时具有“先写后读”和“先读后写”相关，以交换数据为目的

7.6.2 程序段间的相关性分析

相关 执行	数据相关	数据反相关	数据输出相关	相互交换	无相关
顺序串行	√	√	√	×	√
交换串行	有条件	×	×	×	√
并行	×	有条件	有条件	√（完全同步）	√

7.6.3 并行程序设计语言

- 为了加强程序并行性的识别能力，有必要在程序语言中增加能明确表示并发进程的成分，这就是并行程序设计语言。
- 设计并行程序设计语言方法：
  - (1) 重新设计新语言
  - (2) 扩充现有语言功能
  - (3) 不改变现有语言，提供函数库或并行化编译系统

(1) 重新设计新的并行语言

- 可以完全摆脱串行语言的束缚,从语言成分上直接支持并行，这样就可以使并行程序的书写更方便、更自然,相应的并行程序也更容易在并行机上实现。
- **缺点：** 没有统一计算机模型。
- 虽有并行语言，但没有一个被普遍接纳

(2) 扩充现有的串行语言

- 在现有的程序设计语言的基础上扩展出能表示并行进程的语句。
- 若用原来的串行编译器来编译，标注的并行扩充部分将不起作用，仍将该程序作为一般的串行程序处理。
- 若使用扩充后的并行编译器来编译，则该并行编译器就会根据标注的要求，将原来串行执行的部分转化为并行执行。

(3) 提供并行函数库和并行化编译系统

- 为已有的串行语言提供并行运行库。只需要在原来的串行程序中加入对并行库的调用，就可以实现并行程序设计。
  - 如现在流行的MPI（消息传递接口）并行程序设计就属于这种方式。
- 针对以上的方式实现并行语言，一般采用下述集中编译器方法完成并行语言的编译处理：
  - 新语言编译器
  - 预编译处理
  - 并行函数与类库
  - 并行化编译系统

并行程序设计语言关键技术

(1)进程管理

①进程创建与消亡：显式、隐式

显式如FORK-JOIN，隐式如cobegin-coend、parfor等

②进程激活：创建时激活、收到消息时激活

一般采用创建时激活

③进程粒度：一般提供中粒度(2K~10K指令)

进程描述手段(因进程创建代价较大)

并行程序设计语言关键技术

(2)进程通信与同步

通信方式：同步、异步互锁、异步非互锁

通信 — 进程间数据传递，一般采用异步方式(提高性能)

同步 — 进程间协同，一般采用同步方式(提高可靠性)

在多处理机系统中，处理机的数目多少是不会影响程序的编写的，所编写的并行程序可以在机数不同的多处理机系统上通用。

并行程序设计模型

- 3种程序设计模型 (1/3):
  - （1）共享内存模型
    - 多个线程或进程同时运行
    - 它们共享同一内存资源，每个线程或进程都可以访问该内存的任何地方
    - 例如 openMP 就是采用共享内存模型

并行程序设计模型

- 3种程序设计模型 (2/3)：
  - （2）分布式内存模型
    - 多个独立处理结点同时工作，每个处理结点都有一个本地的私有内存空间
    - 进程可以直接访问其私有内存空间。 若一个进程需要访问另一个处理结点处的私有空间，则此进程需要发送信息给那个进程来进行访问
    - MPI 就是采用分布式内存模型

并行程序设计模型

- 3种程序设计模型 (3/3):
  - （3）分布式共享内存模型
    - 整个内存空间被分为共有空间和私有空间
    - 每个线程可以访问所有的共有空间，并且每个线程都有自己独立的私有空间
    - Unified Parallel C 就是采用分割全局地址空间模型

学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.7 多处理机性能

- 使用多处理机的主要目的是为了用多处理机并发执行多个任务来提高解题速度。
- 引起峰值性能下降的原因：
  - 因处理机间通信而产生的延迟
  - 一台处理机与其它处理机同步所需的开销
  - 当没有足够多任务时，一台或多台处理机处于空闲状态
  - 由于一台或多台处理机执行无用的工作
  - 系统控制和操作调度所需开销

7.7 多处理机性能

- 任务粒度的大小会显著影响多处理机的性能和效率。
- 并行性在很大程度上依赖于R/C比值
  - R — 程序用于有效计算的执行时间
  - C — 处理机间通信等辅助开销时间
- 通常：
  - R/C比值小，细粒度并行，并行性低
  - R/C比值大，粗粒度并行，并行性高

为获得最佳性能，应对并行性和额外开销大小进行权衡，也要与应用问题的粒度取得适配。

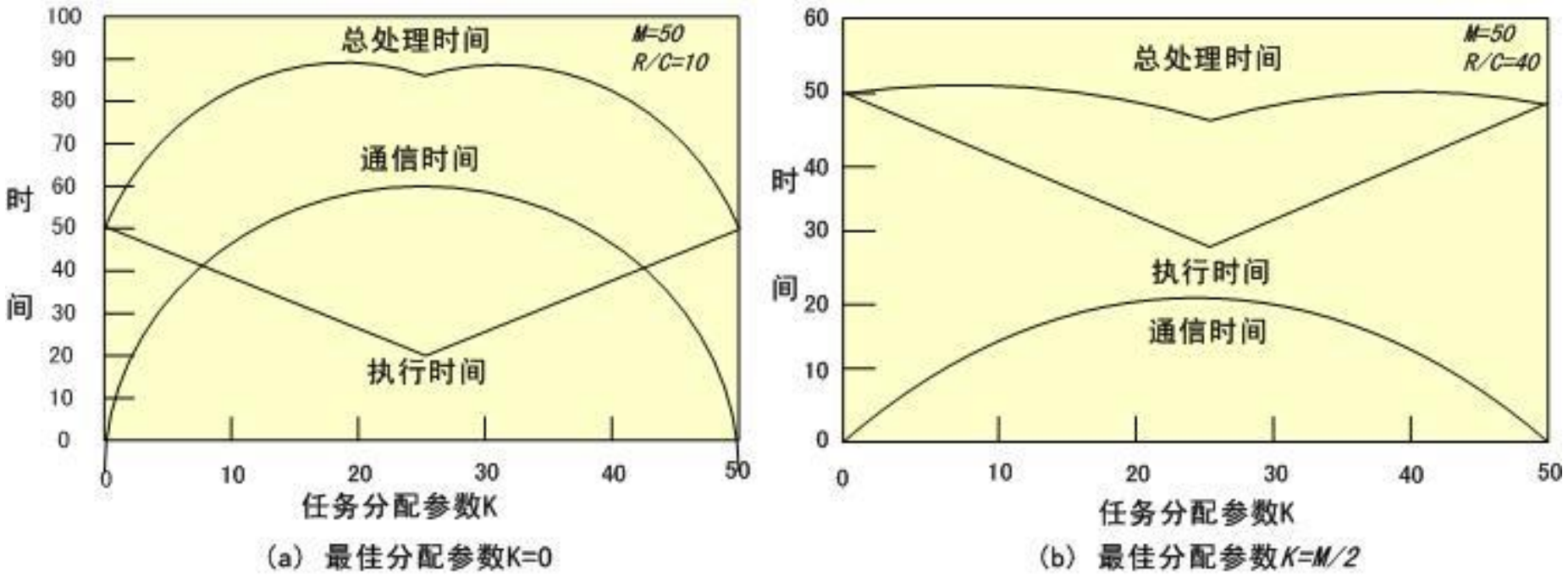
7.7.1 基本性能

- 假设有一个包含 M 个任务的应用程序，希望在一个由 N 台处理机组成的系统上以最快的速度执行这个程序。
- 假设：
  - （1）每个任务的执行时间为R个单位；
  - （2）当两个任务不在同一台处理机上时，其通信所需的额外开销为C个单位时间。当两个任务在同一台处理机上时，通信所需的额外开销为0。

总处理时间＝R•max(M－K，K) ＋ C•(M－K)•K

总处理时间＝执行时间 + 通信和其它额外开销的时间

7.7.1 基本性能



两种不同R/C比值的并行执行时间

对于该模型的结论是：当 R/C ＜ M/2 时，把所有任务分配给同一台处理机能使总处理时间最小；当 R/C ＞ M/2 时，把任务平均地分配给两台处理机能使总处理时间最小。也就是说使任务分配参数 K＝0 或 K＝M/2。

当M为奇数时，应使K尽可能接近 M/2。

7.7.2 N台处理机系统的基本性能

- 我们将第 Ki 个任务分配给第 i 台处理机。基本性能等式可推广为：

$$\text{总处理时间} = R \cdot Max(Ki) + \frac{C}{2}\sum_{i} K(M - Ki)$$

$$= R \cdot Max(Ki) + \frac{C}{2}(M^2 - \sum_{i} Ki^2)$$

为使总处理时间最小：将所有的任务都集中在一台处理机上，或者将任务平均分配给所有处理机。

所谓平均是指如果M是N的倍数，则每台处理机分得M / N个任务，否则除一台处理机外其它处理机分得M / N个任务，那一台处理机分得剩余的任务。

7.7.2 N台处理机系统的基本性能

$$\text{加速比} = \frac{R \cdot M}{\left(\frac{RM}{N} + \frac{CM^2}{2} + \frac{CM}{2N}\right)} = \frac{R}{\frac{R}{N} + \frac{CM\left(1 - 1/N\right)}{2}}$$

$$= \frac{\frac{RN}{C}}{\frac{R}{C} + \frac{M(N-1)}{2}}$$

如果分母中的第一项远远大于第二项，即M，N较小，R / C较大，加速比与N成正比例。如果处理机台数N很大， 则分母主要由第二项决定，加速比与R / CM成正比例，而不依赖于处理机的台数了。

所以处理机的台数不应超过由成本与R / C比值函数所决定的极大值。

学习内容

- 7.1 多处理机概念
- 7.2 多处理机结构
- 7.3 多核处理器
- 7.4 多处理机的多Cache一致性
- 7.5 多处理机的机间互连形式
- 7.6 程序并行性
- 7.6 多处理机性能
- 7.7 多处理机操作系统

7.8 多处理机操作系统

- 由多台计算机协同工作来完成所要求任务的操作系统
  - 负责处理机分配、进程调度、同步和通信、存储系统管理、文件系统与I/O设备管理、故障管理与恢复等
- 按照结构来划分，目前有三种类型：
  - 主从式(Master-slave)
  - 独立监督式(Separate Supervisor)
  - 浮动监督式(Floating Supervisor)

7.8 多处理机操作系统

- 主从式(Master-slave)
  - 由一台主处理机进行系统的集中控制，负责记录、控制其它从处理机的状态，并分配任务给从处理机。
  - 优点：硬件和软件结构相对简单
  - 缺点：对主处理机可靠性要求很高。当不可恢复错误发生时，系统容易崩溃，此时必须重新启动主处理机。系统灵活性差，在控制使用系统资源方面效率也不高。

7.8 多处理机操作系统

- 独立监督式(Separate supervisor) (1/2)
  - 操作系统将控制功能分散给多台处理机，共同完成对整个系统的控制工作。每个处理机均有各自的管理程序(操作系统的内核)。
  - 优点：
    - 每个处理机都有其专用的管理程序，故访问公用表格的冲突较少，阻塞情况自然也就较少，系统的效率较高
    - 每个处理相对独立，因此一台处理机出现故障不会引起整个系统崩溃

7.8 多处理机操作系统

- 独立监督式(Separate supervisor) (2/2)
  - 缺点：
    - 减少了对控制专用处理机的需求，但是实现更复杂
    - 每个管理程序都有一套自用表格，但仍有一些共享表格，从而发生表格访问冲突问题，导致进程调度复杂性和开销的加大
    - 修复故障造成的损害或重新执行故障机未完成的工作非常困难
    - 各处理机负荷的平衡比较困难。

7.8 多处理机操作系统

- **浮动监督式(Floating supervisor)**
  - 系统中每次只有一台处理机作为执行全面管理功能的“主处理机”，“主处理机”可以根据需要浮动，即从一台切换到另一台处理机。这样，即使执行管理功能的主处理机故障，系统也能照样运行下去；
  - 优点：
    - 系统可靠性更强，没有单主处理崩溃瓶颈
    - 更好的平衡处理机负载
  - 缺点：系统实现较复杂

多处理机操作系统的难度

- 处理机的分配和进程调度
- 进程间的同步
- 进程间的通信
- 存储系统的管理
- 文件系统的管理
- 系统重组 等

本章重点

- 多处理机特点及主要技术问题
- 多处理机结构
  - 紧耦合和松耦合
  - UMA、NUMA、ccNUMA和COMA结构及特点
  - MPP、机群(COW/NOW)结构及特点
  - 多核处理器结构及特点
- 多处理机Cache一致性问题及协议
  - 监听协议：写无效与写更新 + 写直达与写回
  - 基于目录的协议

本章重点

- 程序并行性
  - 并行算法及研究思路
  - 多处理机上的相关
  - 并行程序设计模型
- 多处理机的性能
  - 任务粒度与系统性能的关系
  - 多处理机性能的基本模型
- 多处理机的操作系统
  - 3类不同的操作系统的特点及适用场合

第7章 作业7

- 7-2
- 7-6
- 7-7