并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

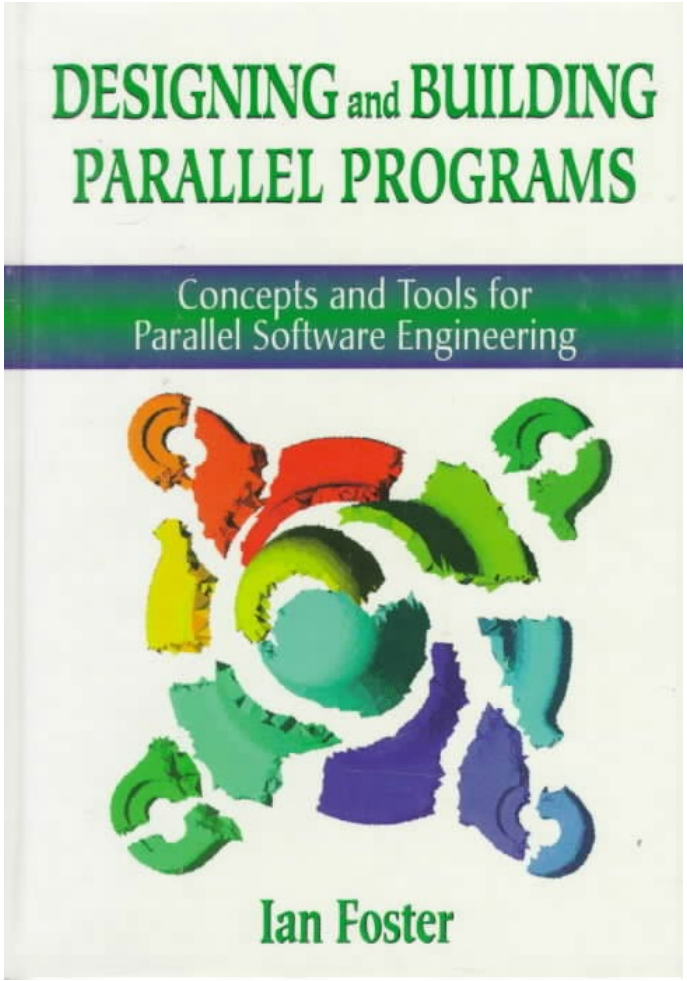
概要

- 第二篇 并行算法的设计
  - 第五章 并行算法与并行计算模型
  - 第六章 并行算法基本设计策略
  - 第七章 并行算法常用设计技术
  - **第八章 并行算法一般设计过程**

第八章 并行算法一般设计过程

- **8.1 PCAM设计方法学**
- 8.2 划分
- 8.3 通信
- 8.4 组合
- 8.5 映射

PCAM设计方法学

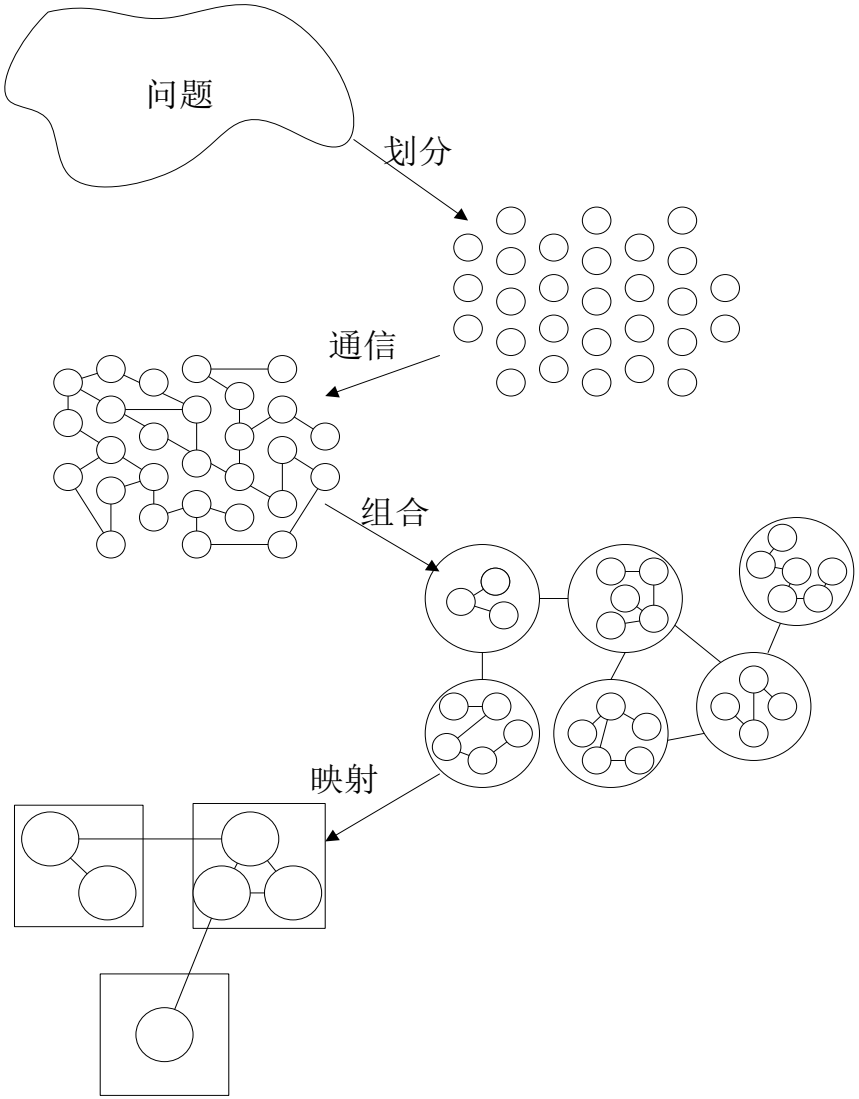


Argonne National Laboratory
University of Chicago

PCAM设计方法学

- 设计并行算法的四个阶段
  - 划分 (Partitioning)
  - 通信 (Communication)
  - 组合 (Agglomeration)
  - 映射 (Mapping)
- 划分：分解成小的任务，开拓并发性；
- 通信：确定诸任务间的数据交换，检测划分的合理性；
- 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设计方法学



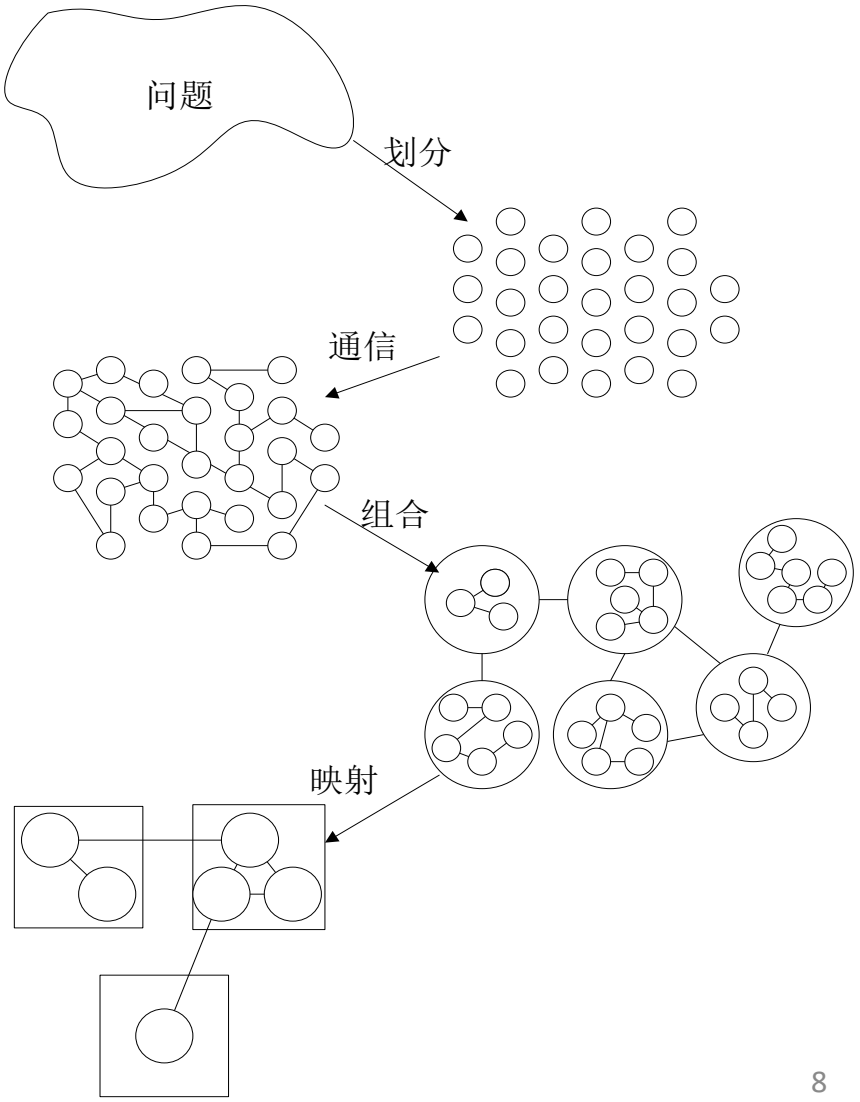
PCAM设计方法学

- 一、二阶段考虑并发性、可扩放性，寻求具有这些特性的并行算法。
  - 即前期主要考虑如并发性等与机器无关的特性。

- 三、四阶段，将注意力放在局部性及其它与性能有关的特性上。
  - 即后期考虑与机器有关的特性。

第八章 并行算法一般设计过程

- 8.1 PCAM设计方法学
- **8.2 划分**
  - 8.2.1 方法描述
  - 8.2.2 域分解
  - 8.2.3 功能分解
- 8.3 通信
- 8.4 组合
- 8.5 映射



划分方法描述

- 充分开拓算法的并发性和可扩放性;
- 划分阶段忽略处理器数目和目标机器的体系结构;
- 分为两类划分:
  - 域分解(domain decomposition)
  - 功能分解(functional decomposition)
- 先考虑数据分解(称域分解)，再考虑计算功能的分解(称功能分解);
- 使数据集和计算集互不相交（？？？）

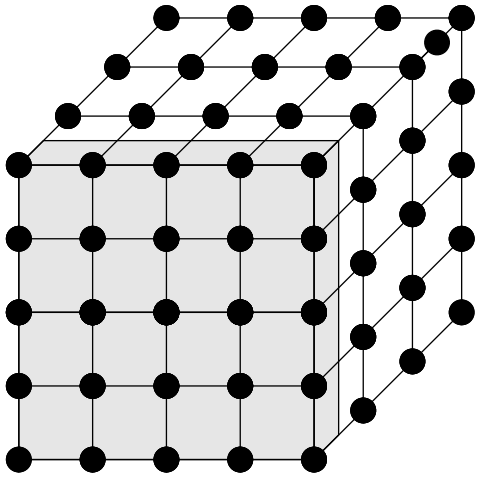
partition both computation and data into disjoint sets
将计算和数据都划分为一些不相交集合
《Designing and Building of Parallel Algorithms》

域分解

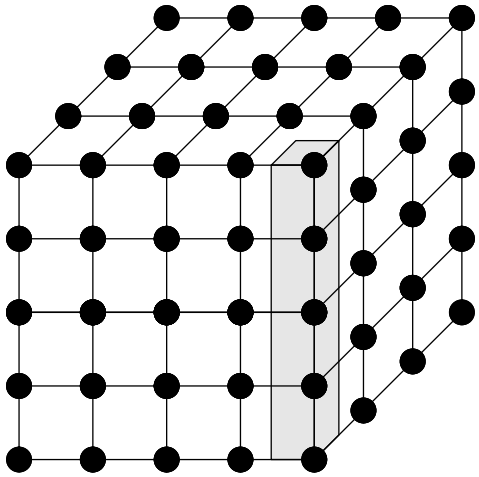
- 划分的对象是数据，可以是算法的输入数据、中间处理数据和输出数据;
- 将数据分解成大致相等的小数据片;
- 划分时考虑数据上的相应操作;
- 如果一个任务需要别的任务中的数据，则会产生任务间的通信;

域分解

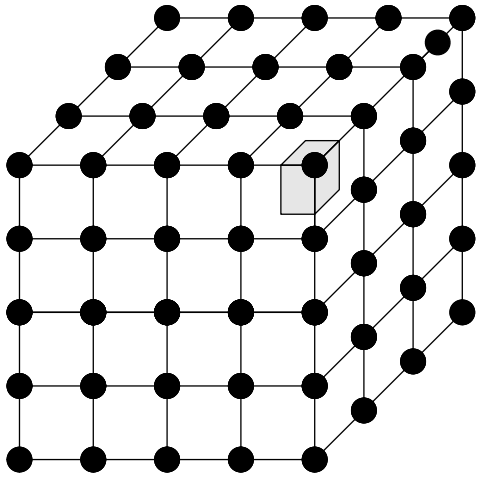
- 示例：三维网格的域分解，各格点上计算都是重复的。下图是三种分解方法：



1-D



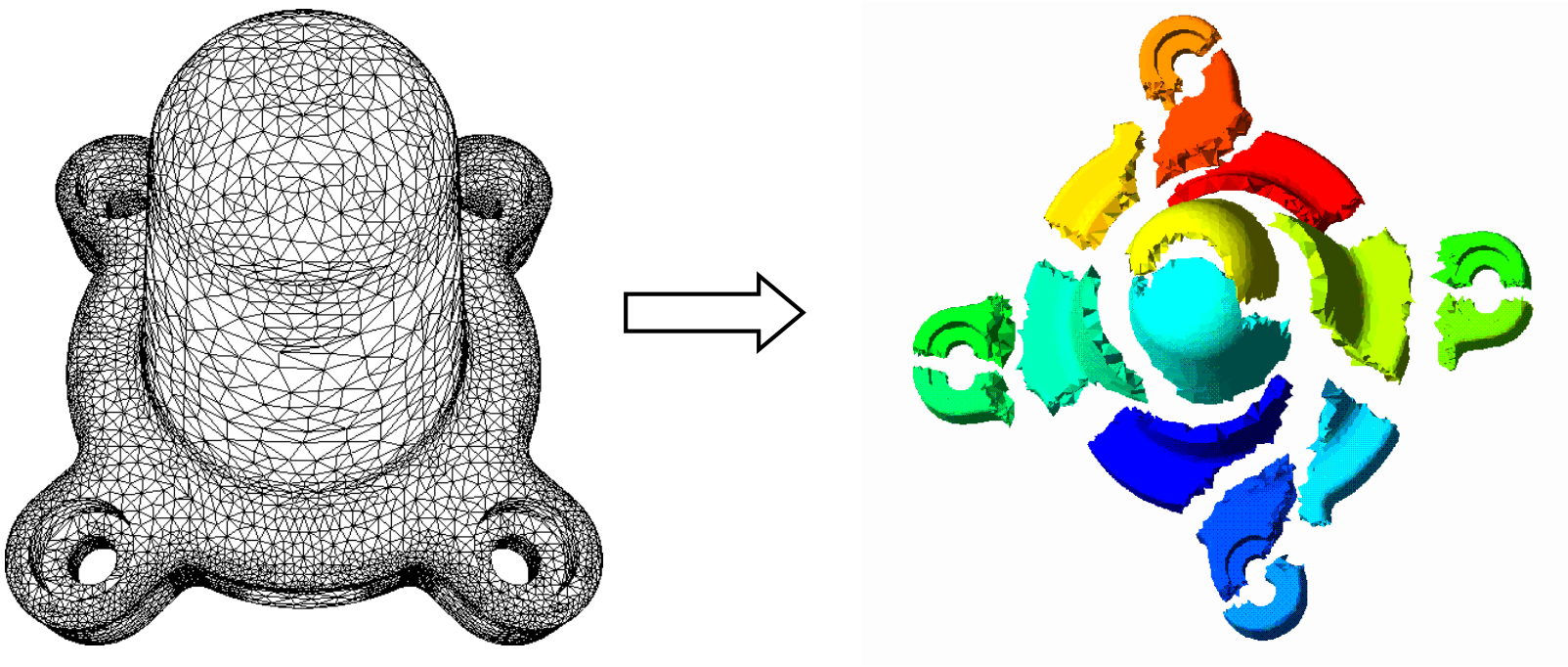
2-D



3-D

域分解

- 不规则区域的分解示例:



域分解

工程上的流动与传热问题大多发生在复杂区域内

网格生成：计算流体和传热中重要的研究领域

数值计算的最终精度及效率，取决于：

- 生成的网格
- 采用的算法

高性能数值计算： 网格生成，求解算法 良好匹配

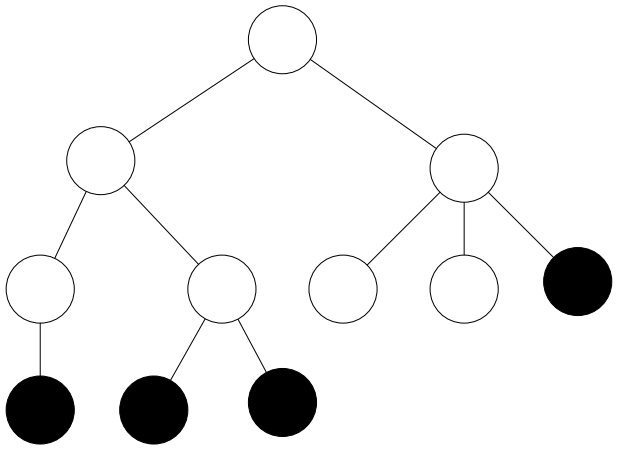
功能分解

- 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；

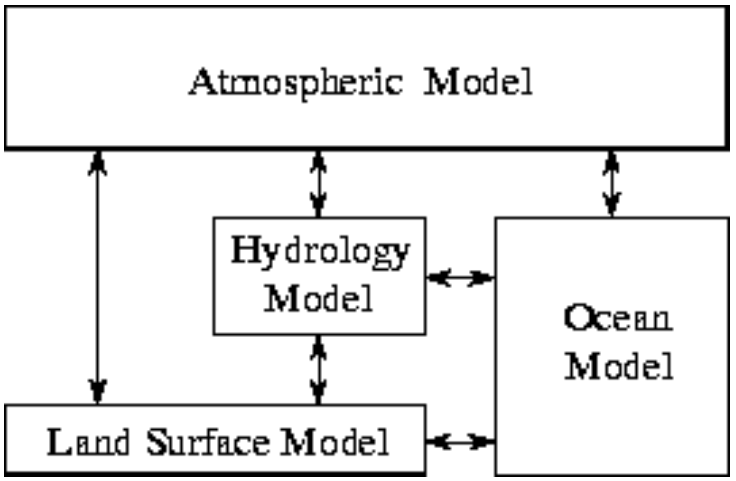
- 划分后，研究不同任务所需的数据。如果这些数据不相交的，则划分是成功的；如果数据有相当的重叠， 意味着要重新进行域分解和功能分解；
- 功能分解是一种更深层次的分解。

功能分解

- 示例1：搜索树



- 示例2：气候模型

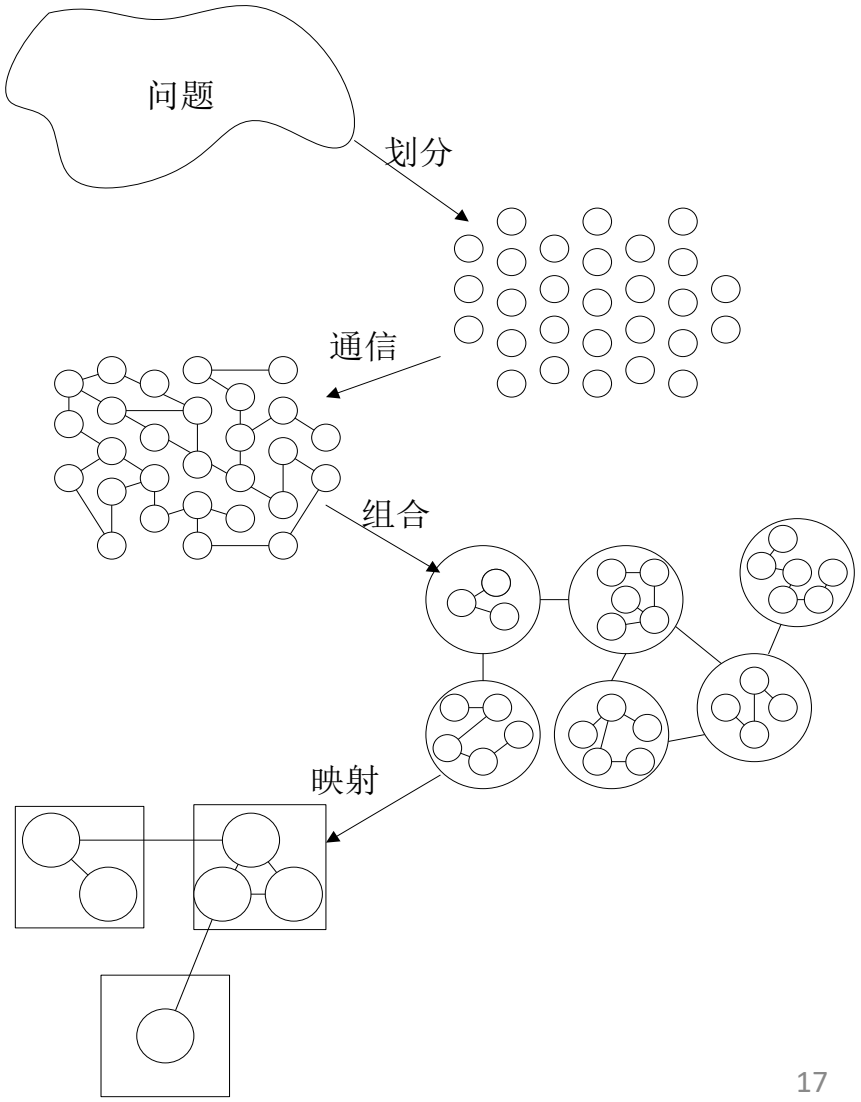


划分checklist

- 划分是否保持映射和扩展的灵活性?
- 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?
- 是否有其他划分方案可以考虑?

第八章 并行算法一般设计过程

- 8.1 PCAM设计方法学
- 8.2 划分
- **8.3 通信**
  - 8.3.1 方法描述
  - 8.3.2 四种通信模式
- 8.4 组合
- 8.5 映射

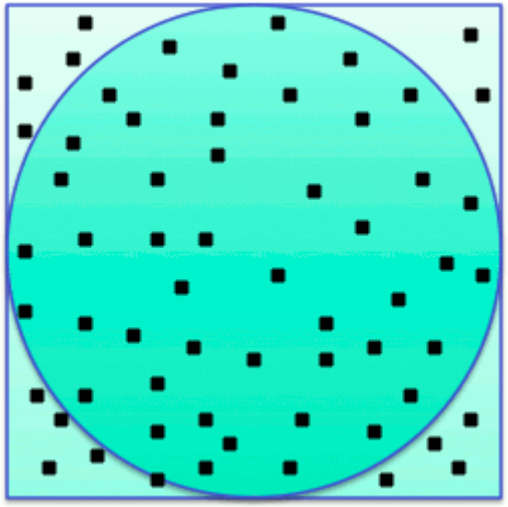


通信方法描述

- 通信是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；从而产生了通信；
- 诸任务是并发执行的，通信则限制了这种并发性；

Embarrassingly parallel

- 在并行计算中，Embarrassingly parallel（易并行）指的是划分后几乎不需要通信的可并行任务
- 并行算法的理想情况
- 案例：Monte Carlo Simulation



```
n = N/n_proc;  // assume N is a multiple of n_proc
Broadcast(n, 0);
my_count = Count_pt(n);
if (proc_id == 0)
   for (i=1; i < no_proc; i++) {
      Recv(count, i);
      my_count += count;
   }
else
   Send(my_count, 0);
return 4*my_count/N;
```

主要耗时

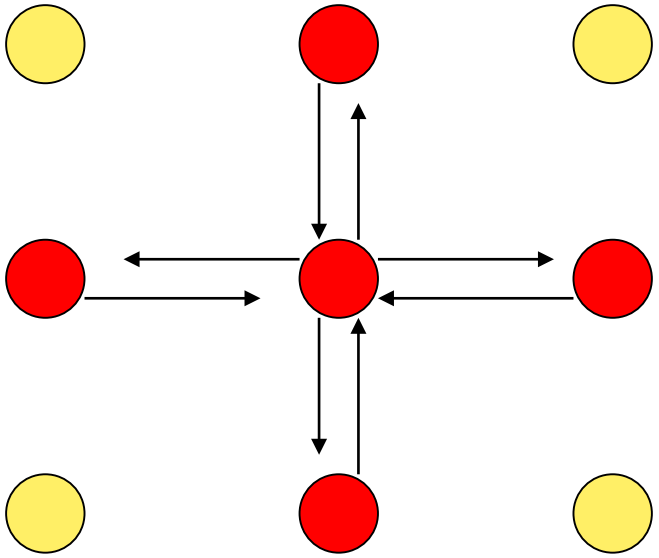
```
Count_pt(int n) {
   count_in = 0;
   for (i=0; i < n; i++) {
      x = randr(-1, 1);
      y = randr(-1, 1);
      if (x*x+y*y <= 1)
         count_in++;
   }
   return count_in;
}
```

四种通信模式

- 局部/全局通信
  - 通信涉及的子任务范围
- 结构化/非结构化通信
  - 通信模式是否规则（收发者ID具有简单关系）
- 静态/动态通信
  - 通信模式是否随时间变化
- 同步/异步通信
  - 通信是否阻塞

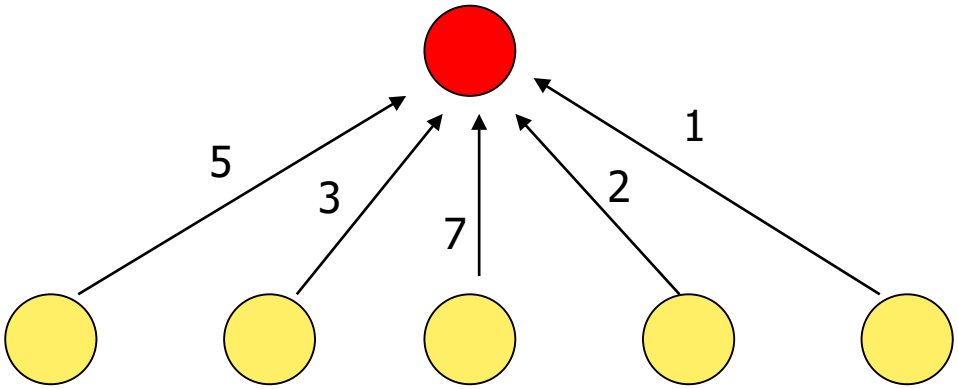
局部通信

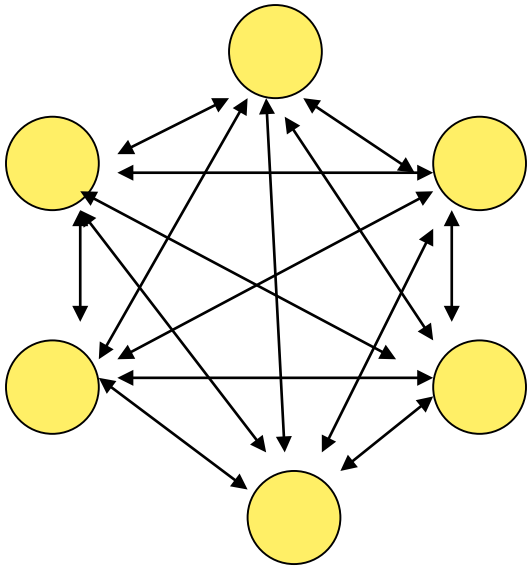
- 通信限制在一个邻域内



全局通信

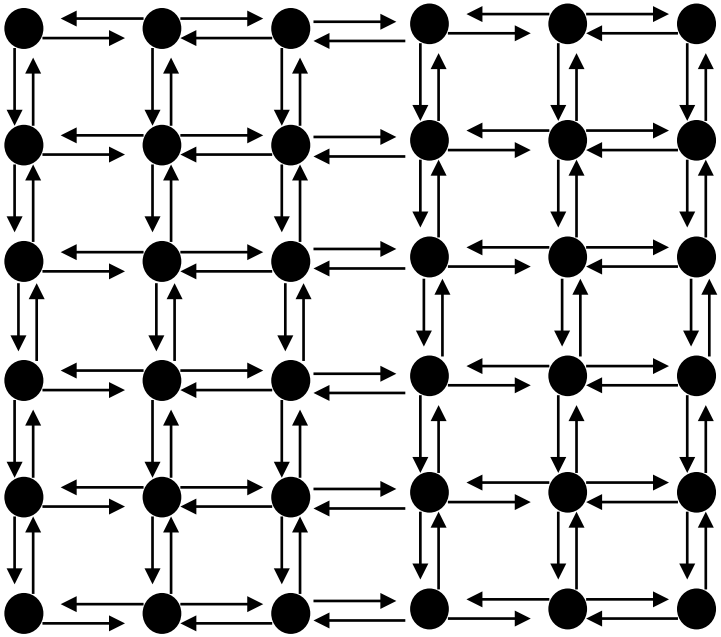
- 多个子任务需要参与交换数据
- 例如：
  - All to All
  - Server-Worker

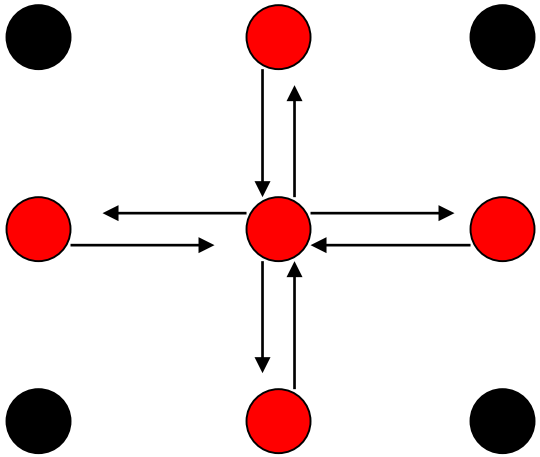




结构化通信

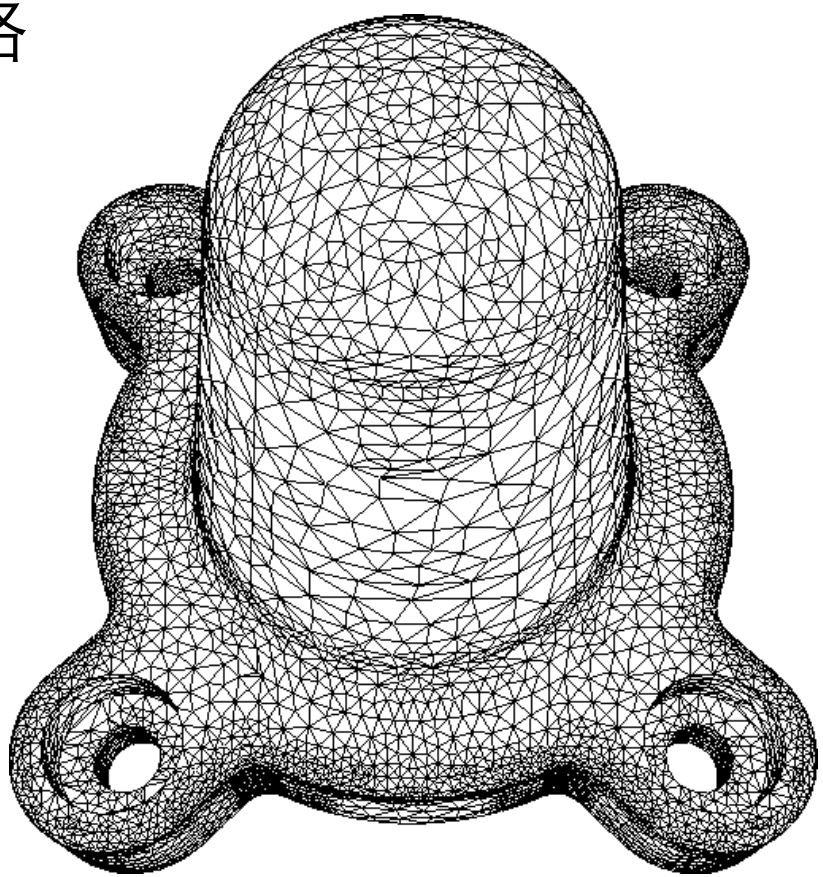
- 每个任务的通信模式是相同的;
- 下面是否存在一个相同通信模式?





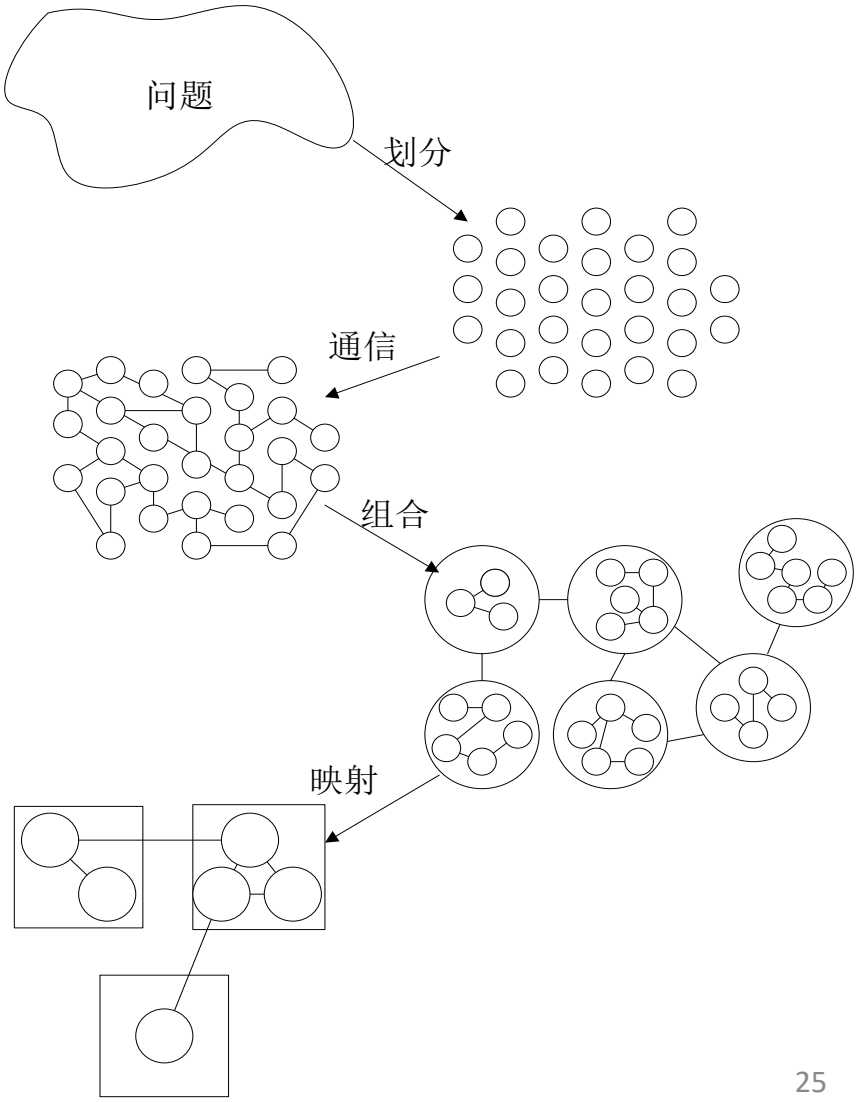
非结构化通信

- 没有一个统一的通信模式
- 例如：无结构化网格



第八章 并行算法一般设计过程

- 8.1 PCAM设计方法学
- 8.2 划分
- 8.3 通信
- **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
- 8.5 映射



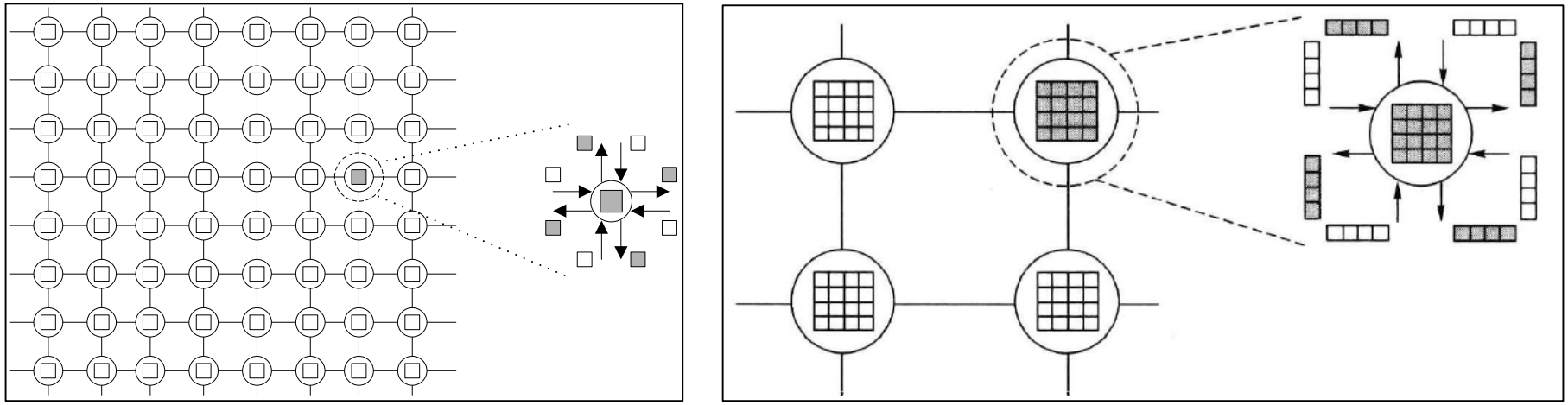
组合方法描述

- 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，可以减少通信成本;
- 保持映射和扩展的灵活性，降低软件工程成本;

粒度（granularity）：单个任务的复杂度，可以用计算量、指令数、串行耗时等指标衡量，是一个不严谨的相对概念（相对于通信等额外开销而言）

表面-容积效应

- 通信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；增大粒度可以减少通信



8x8二维网格的1x1细粒度组合和4x4粗粒度组合

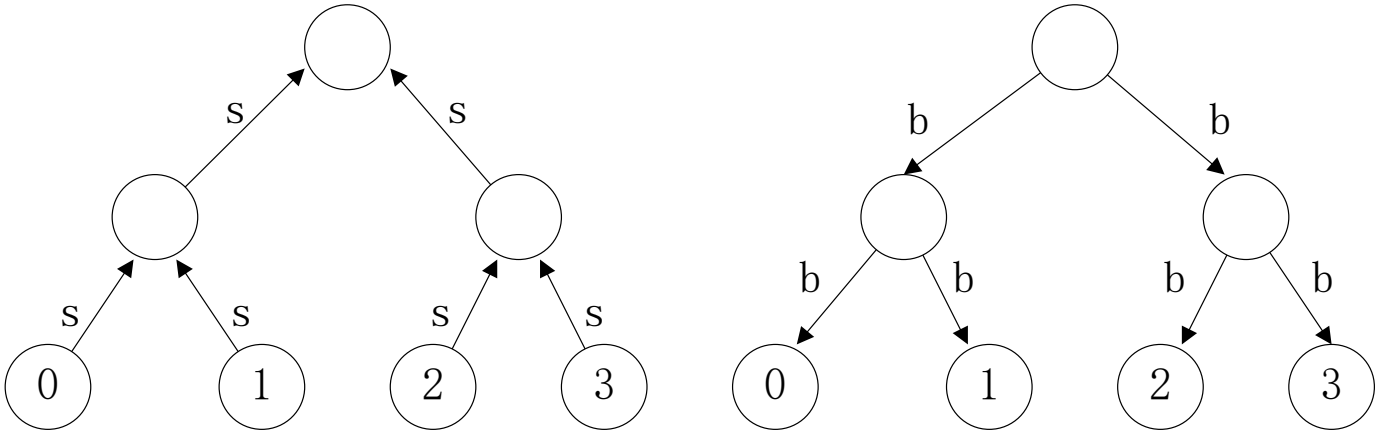
重复计算

- 重复计算减少通讯量，增加计算量，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

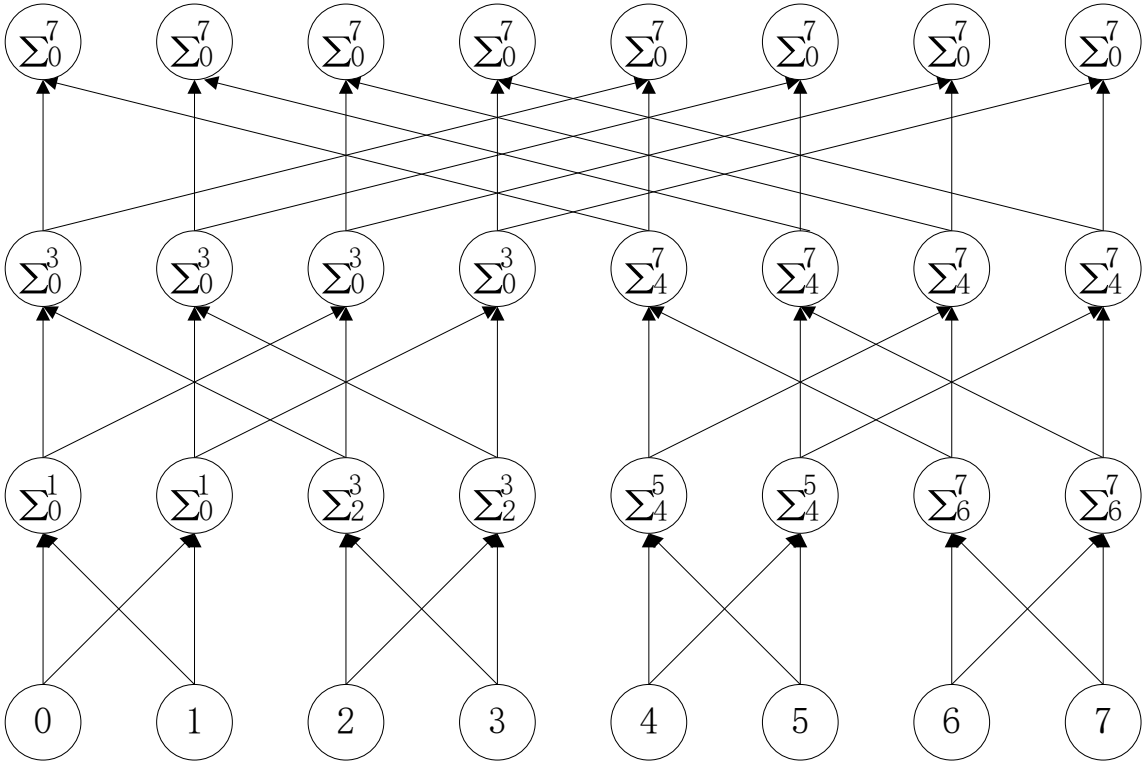
要求每个处理器均保持全和



二叉树上先求和，再广播，共需2logN步通信

重复计算

- 示例（续）：使用重复计算，可以减少通信步



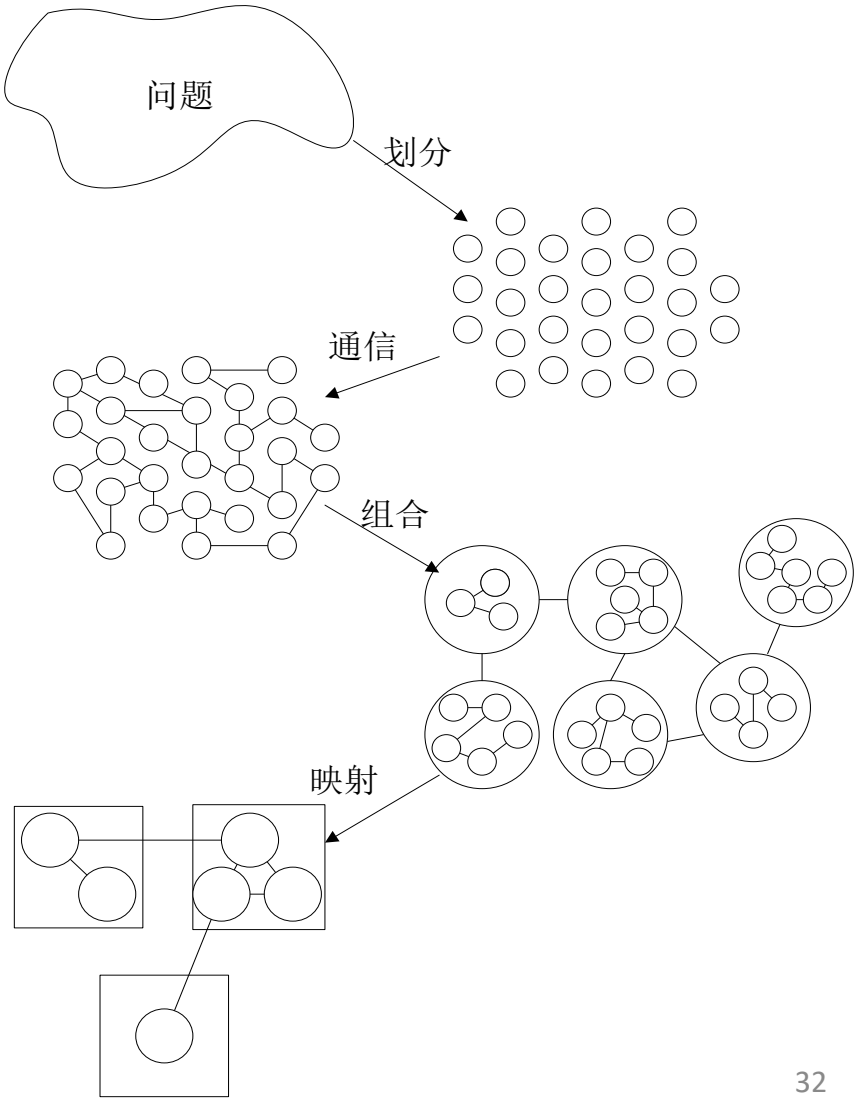
蝶式结构求和，共需logN步

组合checklist

- 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?

第八章 并行算法一般设计过程

- 8.1 PCAM设计方法学
- 8.2 划分
- 8.3 通信
- 8.4 组合
- **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



映射方法描述

- 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
  - 可并发的任务 → 不同的处理器
  - 任务之间存在大量通信量 → 同一处理器
- 映射需要权衡，最优映射属于NP完全问题；

负载平衡

- 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：教材P221
  - 递归对剖
  - 局部算法
  - 概率方法
  - 循环映射

任务调度

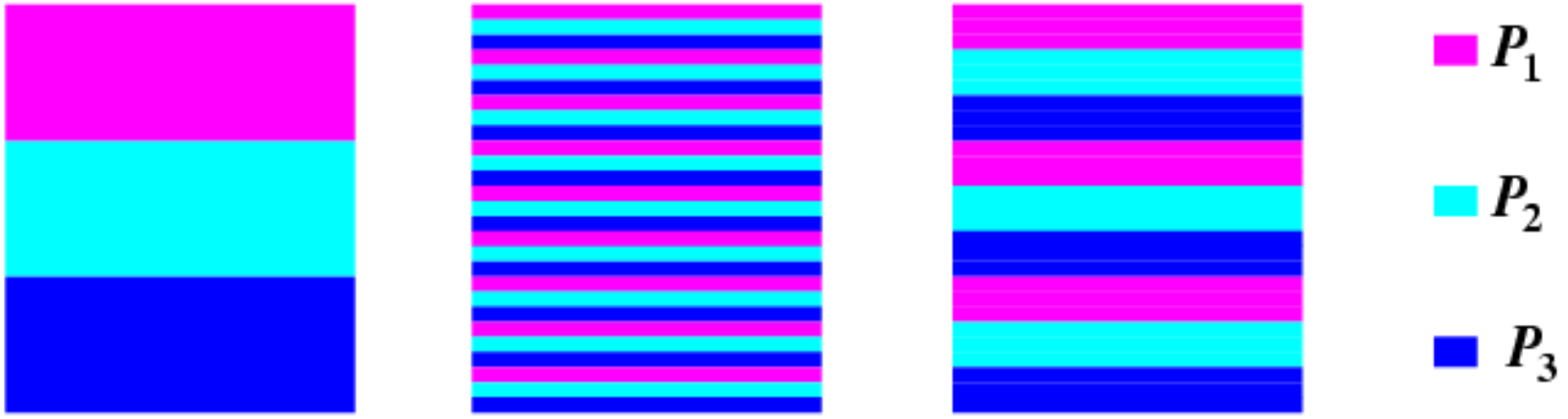
- 负载平衡与任务调度密切相关
- 任务调度通常有静态的和动态的两种方法

任务调度

- 静态调度 (Static Scheduling): 任务到处理器的算术映射
- 静态调度的优点是没有运行时任务管理的开销，但为了实现负载平衡，要求不同任务的工作量和处理器的性能是可以预测的并且拥有足够的可供分配的任务。

任务调度

- 常见的静态调度为每个处理器分配个连续的n/p个循环迭代，其中n为总迭代次数，p是处理器数。
- 也可以采用轮转（Round-robin）的方式来给处理器分配任务，即将第i个循环迭代分配给第i mod p个处理器。



Striped row-major mapping of a 27×27 matrix on $p = 3$ processors.

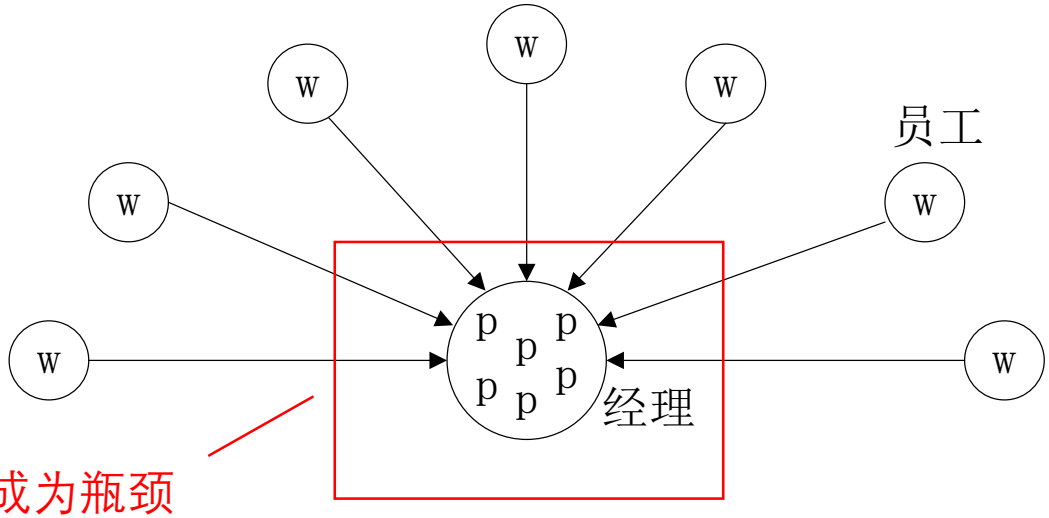
任务调度

- 动态调度(Dynamic Scheduling)相对灵活，可以运行时在不同处理器间动态地进行负载的调整。
- 和静态调度相反，动态调度不需要预先知道各个任务的工作量以及各个处理器的性能，但是会带来运行时任务管理的开销。

任务调度

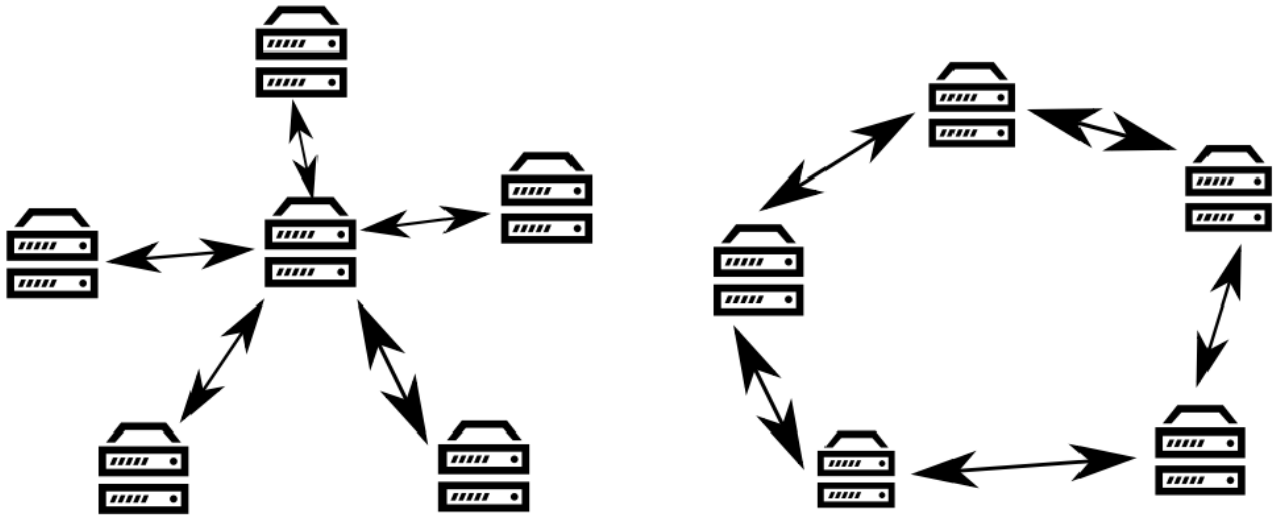
- 经理/雇员模式任务调度

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。



任务调度

- 任务调度案例：分布式深度学习训练

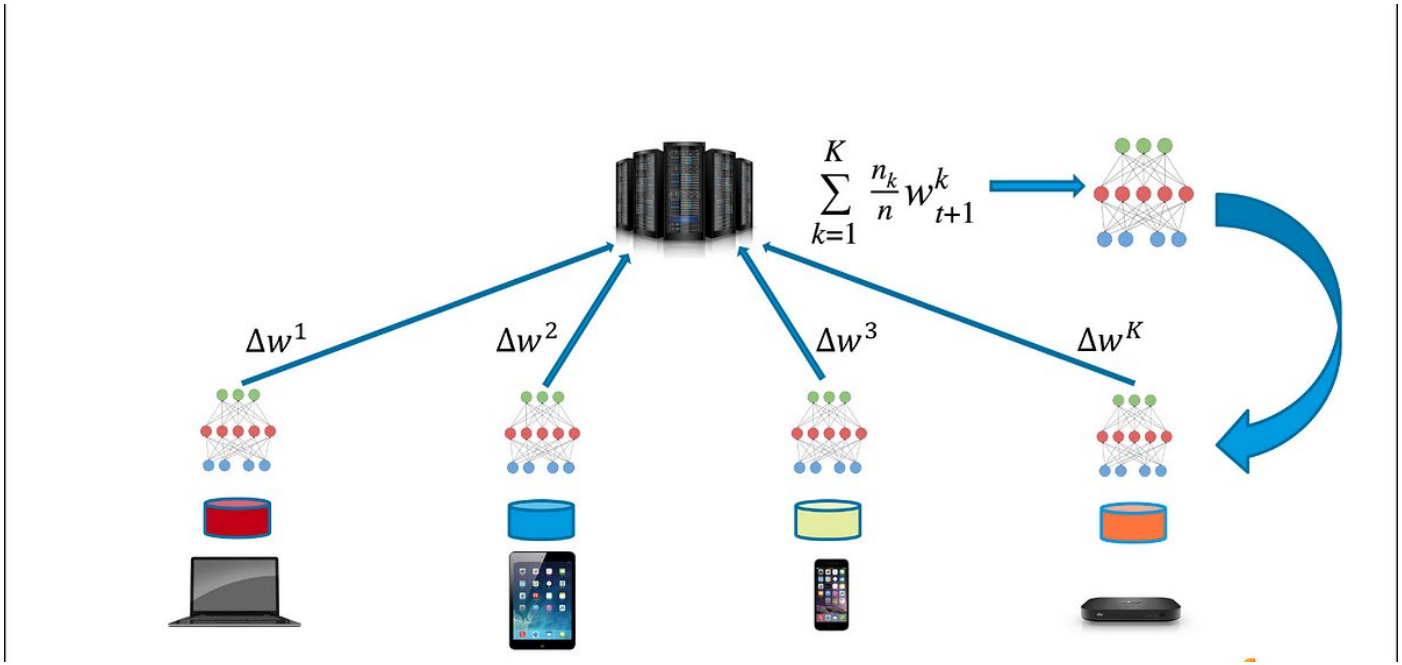


(a) Parameter Server　　(b) Decentralized Architecture

动态调度：参数服务器　静态调度：去中心架构

任务调度

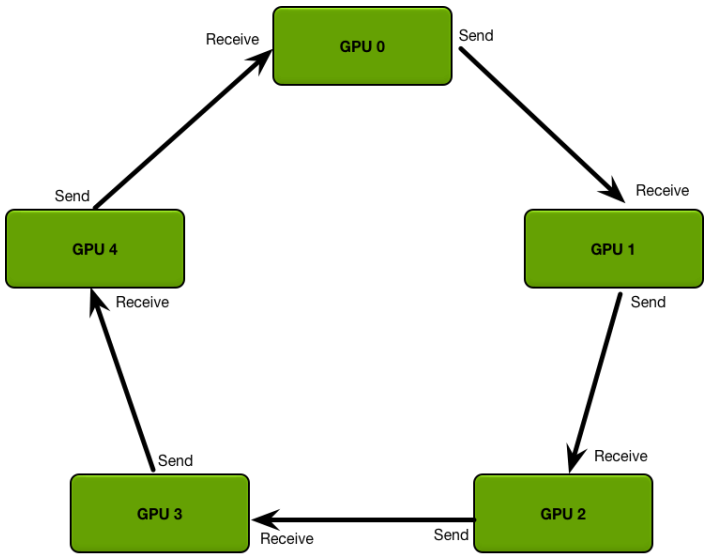
- 动态调度：参数服务器
  - Worker计算能力可能不一样，结果也可能不同时产生

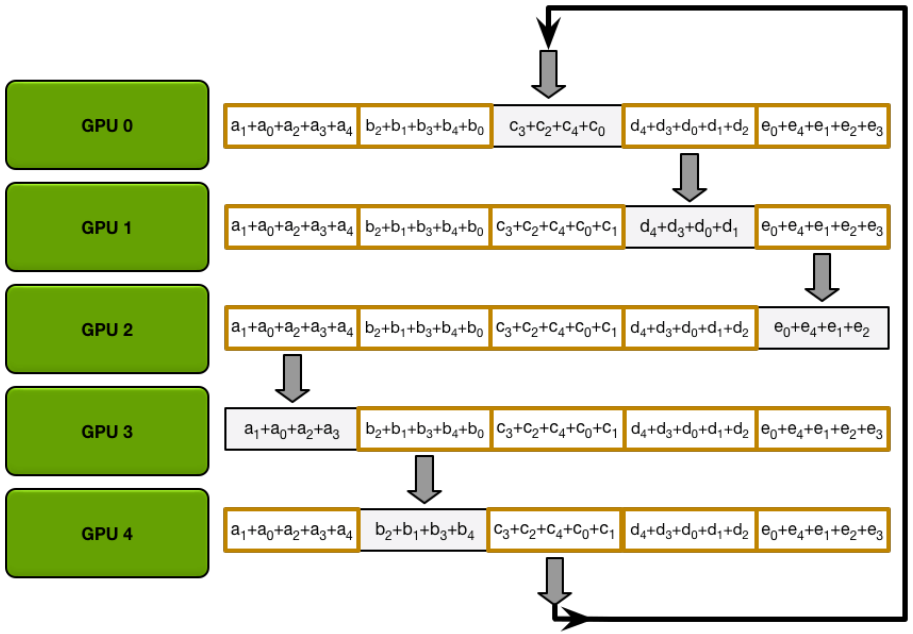


典型应用场景：联邦学习

任务调度

- 静态调度：去中心架构
  - 没有中心服务器成为瓶颈
  - 每个worker需要同步地完成等量任务（Ring-Allreduce算法）





https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

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率

- 映射

  将任务分配到处理器，并保持负载平衡