

第三讲 并行程序设计



目录

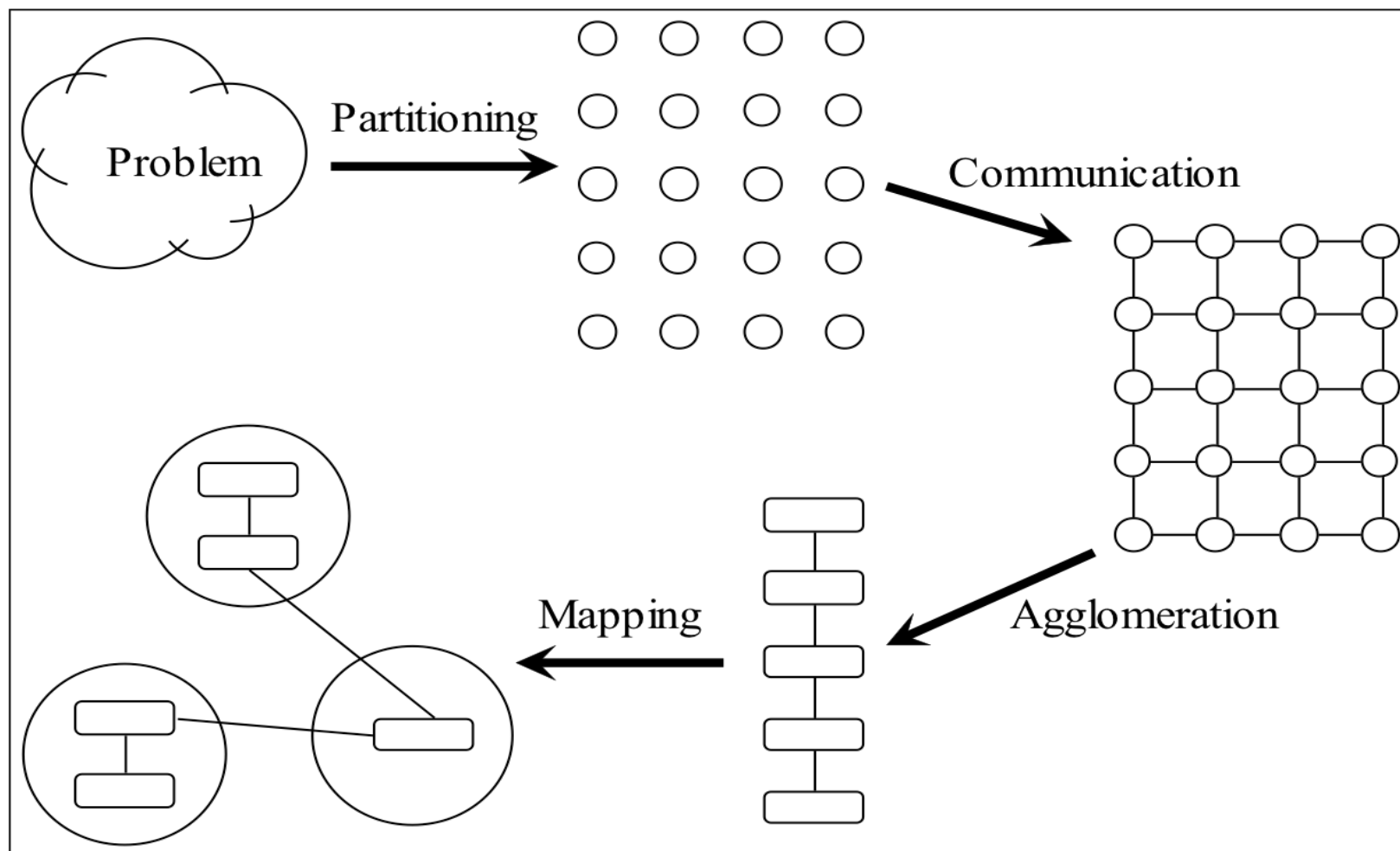
- 算法设计方法论
- 案例研究

Foster 设计方法论 (PCAM)

- 并行算法设计的四个阶段：
 - 划分 (Partition)
 - 通信 (Communication)
 - 聚合 (Agglomeration)
 - 映射 (Mapping)
- 划分和通信：处理与机器无关的问题，影响并发性和可伸缩性。
- 聚合和映射：处理与机器相关的问题，影响局部性和其他性能问题。



Foster 设计方法论



划分

- **阶段1. 划分**：将要执行的**计算**和计算操作的**数据**划分成小任务。
- 尽可能多地确定可以并行执行的任务数量。
- 两种方法：
 - 域分解
 - 将数据分成多个部分
 - 确定如何将计算与数据关联起来
 - 功能分解
 - 将计算分成多个部分
 - 确定如何将数据与计算关联起来



域分解

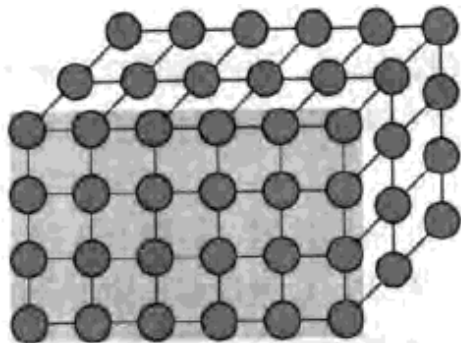
- 将数据尽可能地划分为大小相等的小块。
- 生成多个任务，每个任务都有一些数据和一组对数据的操作。
- 一个好的经验法则是先关注最大的数据结构或访问最频繁的数据结构。



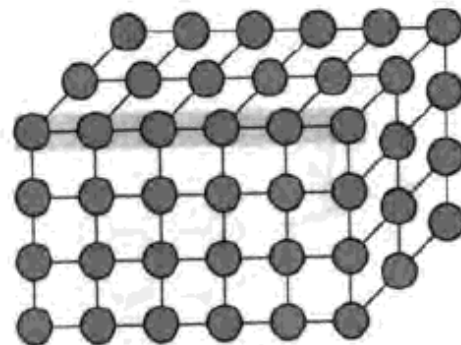
示例

- 对三维模型进行一维、二维或三维分解

数据分解



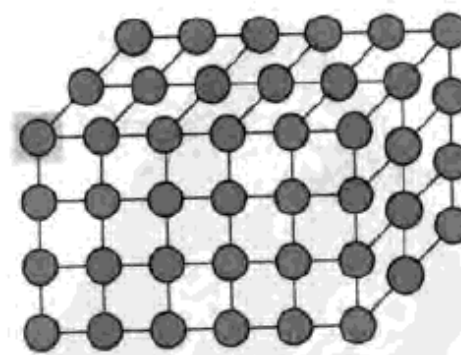
1-D



2-D

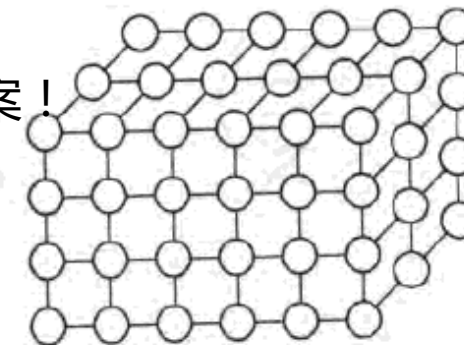


基本任务



3-D

最佳方案！

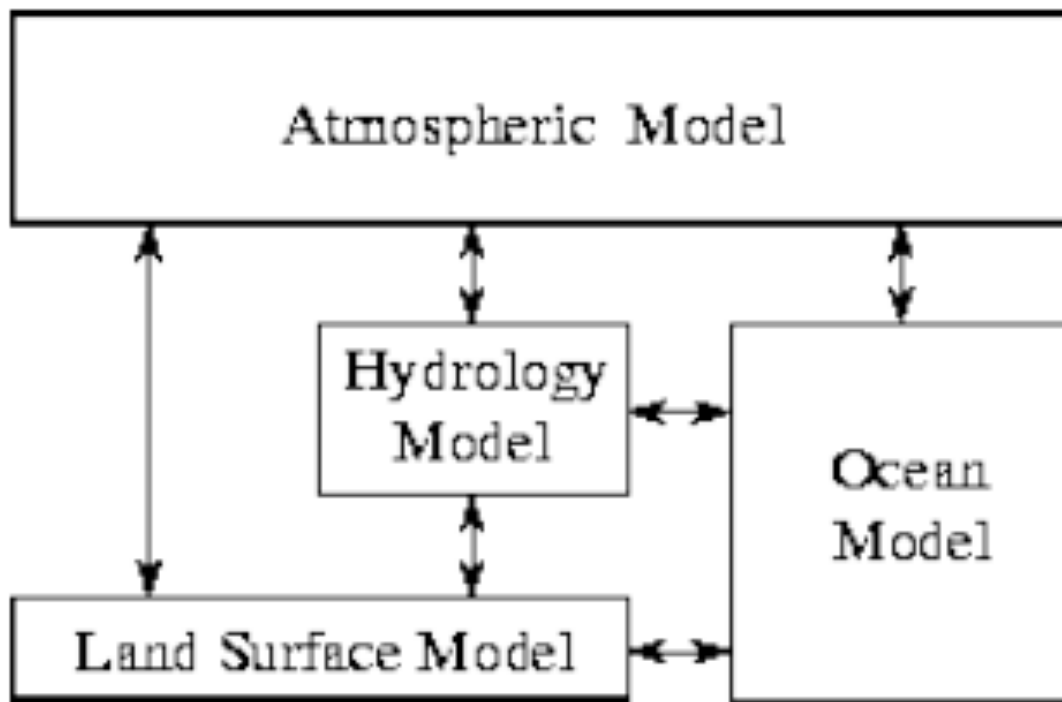


功能分解

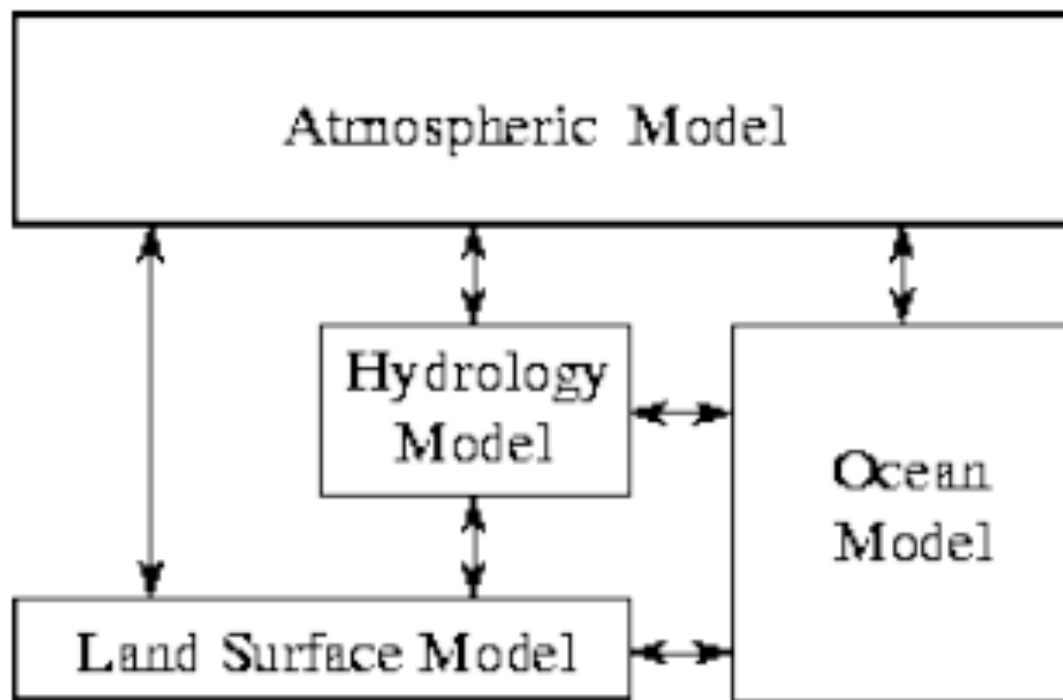
- 首要关注的是需要执行的计算，而不是数据。
- 功能分解是域分解的互补策略。
- 这些任务的数据需求可能是不重叠的，也可能有很大的重叠。
- 功能分解通常通过流水线技术实现并发处理。

功能分解

- 功能分解在程序结构技术中也有着重要的作用
 - 通常适用于复杂系统的计算机模型，这些模型可以被结构化为通过接口连接的简单模型集合。



域分解 vs. 功能分解



- 虽然每个组件可以自然地使用域分解技术进行并行化，但如果首先使用功能分解技术对系统进行分解，则整个并行算法会更加简单。



问题清单：划分

- 划分是否定义了目标并行计算机处理器数量至少一个数量级以上的任务？
- 划分是否避免了冗余的计算和存储需求？
- 任务大小是否大致相同？
- 任务数量是否是问题规模的递增函数？
- 是否确定了几种可供选择的备选划分方式？

为什么我们的目标是尽可能多地确定基本任务？



提问



为什么我们的目标是尽可能多地确定基本任务？

- 因为这能使我们充分考虑并行执行的机会。

通信

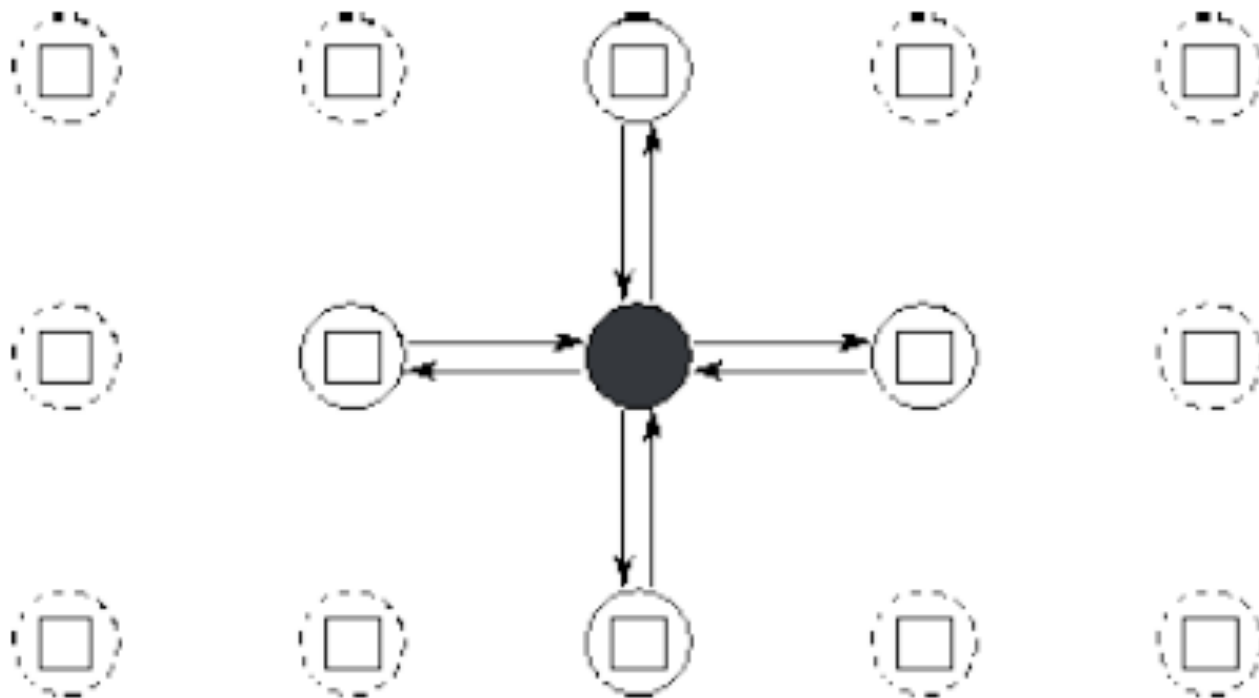
- **阶段2. 通信**：确定在前一步骤中确定的任务之间需要进行哪些通信。
 - 划分产生的任务旨在并发执行，但不是独立执行。
 - 执行任务A的计算需要任务B的数据。
 - 信息流在通信阶段中指定。
 - 我们可以将两个任务之间通信的需求概念化为连接任务的通道（消息传递）。
- 对于功能分解来说，这是很容易的，但对于域分解来说却很困难。

为什么??



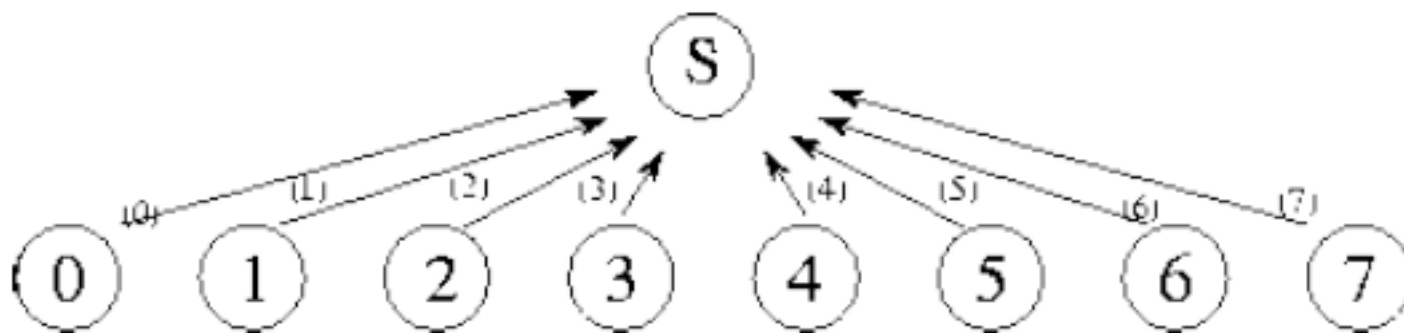
通信模式

- 本地通信
 - 任务需要从少量邻近的任务获取值
 - 创建展示数据流的通道



通信模式

- 全局通信
 - 通常在设计的早期不为它们创建通道
 - 可能会导致过多的通信或限制并发执行的机会。



使用中央管理任务 (S) 汇总分散在N个任务之间的N个部分和。

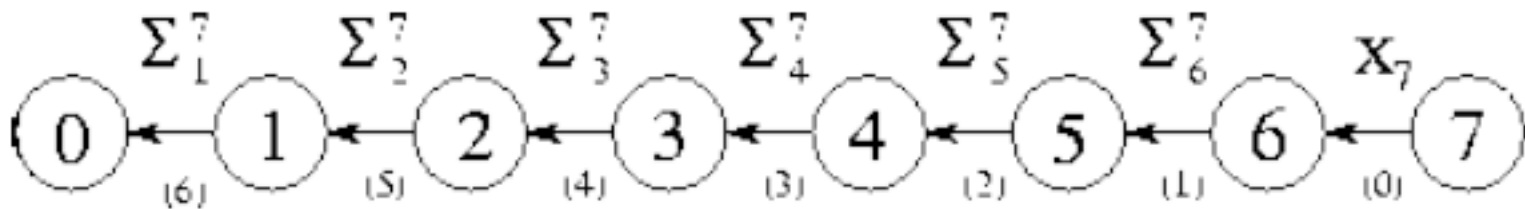


通信模式

- 分发通信和计算

- 分发N个数字的总和，每个任务i计算前i项和：

$$S_i = X_i + S_{i-1}$$



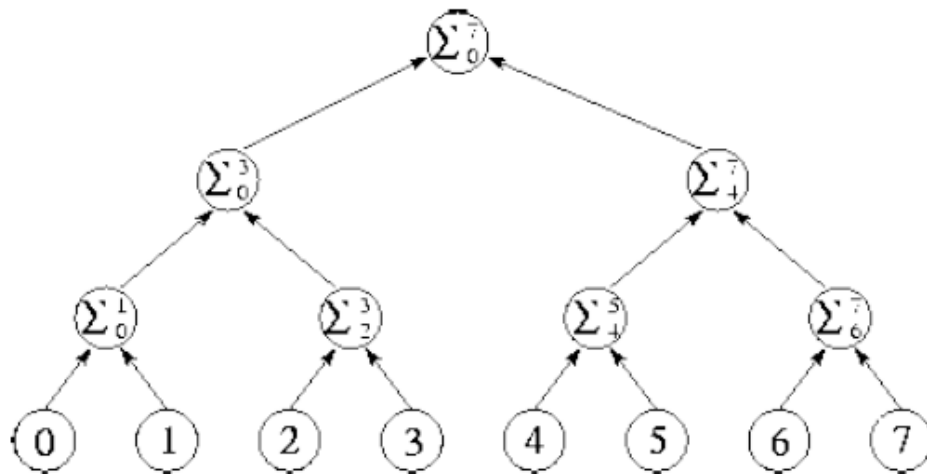
- 将N个任务连接成一个数组，以便对分布在这些任务中的N个数字进行求和。
 - 每个通道都标有步骤编号以及在其上传达的值。
 - 单个求和仍然需要N-1步，但如果有多组求和运算，则可以实现并行（流水线）。



通信模式

- 分治法

- 为了解决一个复杂的问题，将其划分为两个或更多大致相等大小的简单问题。递归应用此过程以产生一组不能进一步细分的子问题。
- 当问题划分生成的子问题可以并发地解决时，分治法在并行计算中非常有效



- 在 $\log N$ 步后，完整的总和在树的根节点得到。
- 一种正则通信结构，每个任务与一小组邻居进行通信。



问题清单：通信

- 通信操作在任务之间是否平衡？
- 每个任务是否与少数邻居进行通信？
- 通信操作是否能够并发执行？
- 不同任务的计算是否能够并发执行？

当前我们希望拥有的，以及我们没有的

- 首先两个步骤都是在寻找问题中的并行性。
- 然而，大量并行任务并不能产生高效的并行算法。此时所获得的设计可能无法很好地映射到真实计算机上。
- 如果任务数量大大超过处理器数量怎么办？
- 现在我们必须确定我们的目标计算机类型，
 - 是集中式多处理器还是多计算机？
 - 我们如何组合这些任务以便将其有效地映射到处理器上？

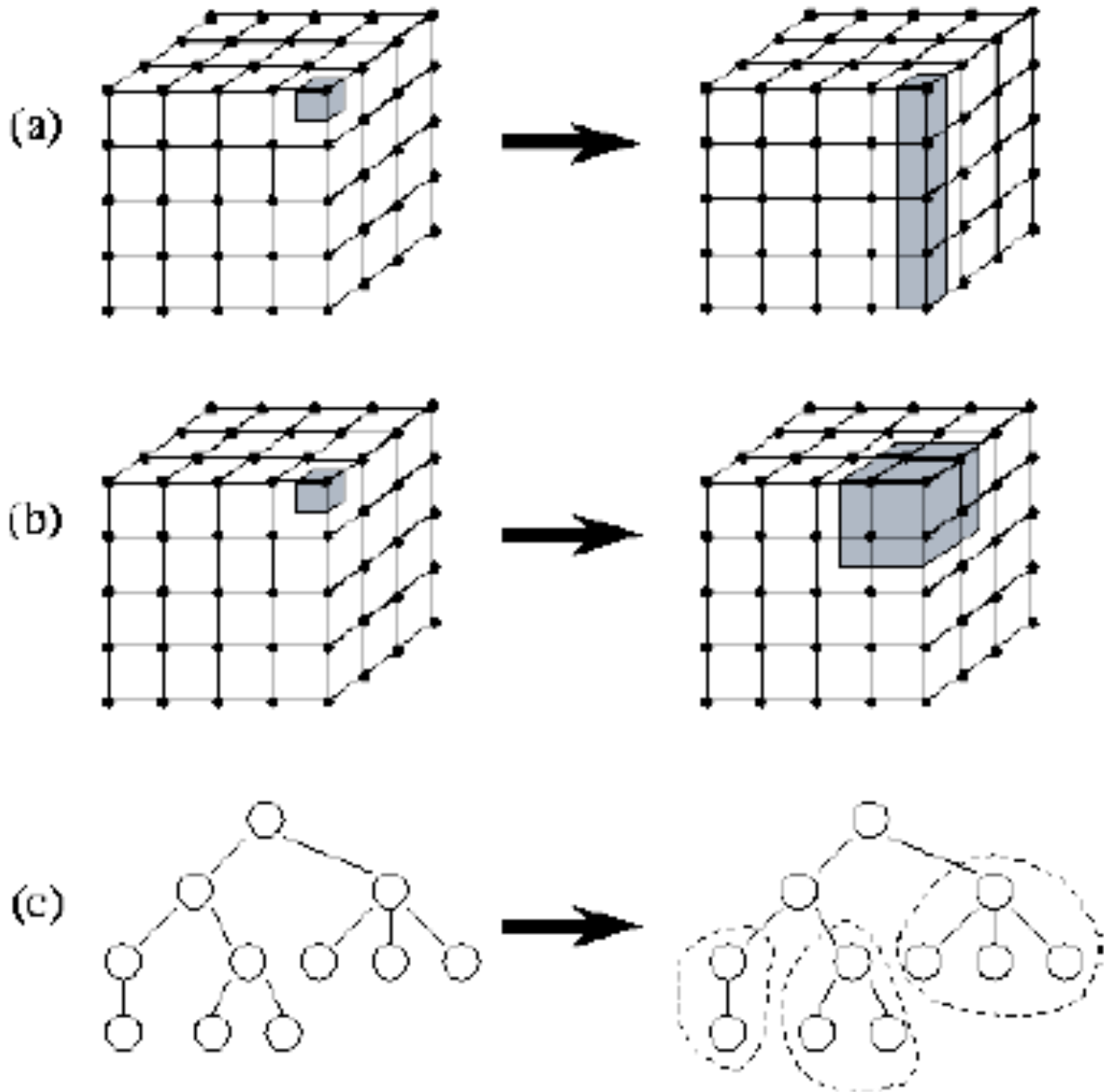


聚合

- **阶段3. 聚合或合并**：将前两个步骤中的任务和通信组合成更大的任务。
 - 举例，如果必须在执行任务B之前执行任务A，则将它们聚合成单个复合任务可能是有意义的。
- 在聚合过程中，需考虑是否值得组合，并确定是否值得复制数据和/或计算。
 - 改善性能
 - 保持程序的可伸缩性
 - 简化编程



示例

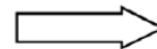
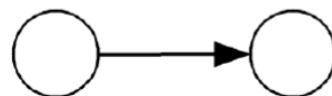


聚合可以改善性能

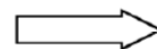
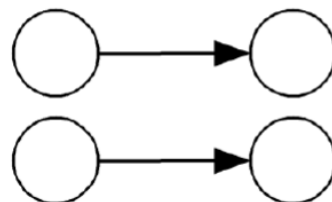
- 降低通信成本

- 将通过通道连接的任务组合在一起可以消除通信增加并行算法的局部性。

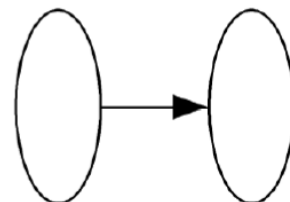
(如图a)



(a)



(b)



- 将发送和接收任务组合成组可以减少消息传输次数。（如图b）

- 减少任务创建成本和任务调度成本。



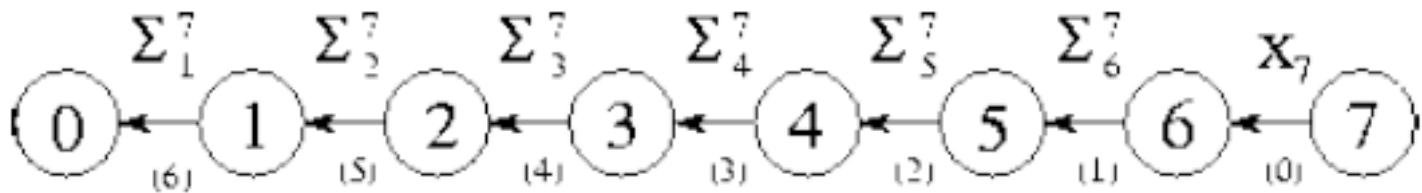
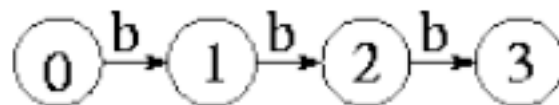
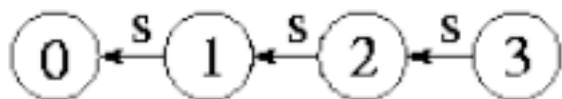
可伸缩性问题

- 假设我们正在操作一个大小为 $8 \times 128 \times 256$ 的三维矩阵，
 - 并且我们的目标机器是具有4个CPU的集中式多处理器。
- 问题？
- Q1：如果我们将第2和第3维聚合起来，我们能在目标计算机上运行吗？
- Q2：如果我们更改为具有8个CPU的集中式多处理器作为目标计算机，我们是否仍然可以在目标计算机上运行？
- Q3：如果我们使用超过8个CPU怎么办？



示例

- 考虑求和问题的变体。求和并确保每个节点都拥有最终结果。



在 $2(N-1)$ 步之后， N 个值的总和会在每个任务中复制。



示例

- 替代算法，任务连接在一个环中
 - 所有 N 个节点都执行相同的算法，并行执行。
 - 经过 $N-1$ 步之后，完整的总和会在每个节点中复制。
- 优点：
 - 在更少的经过时间内执行。
- 缺点：
 - 需要进行不必要（重复）的计算和通信。

问题清单：聚合

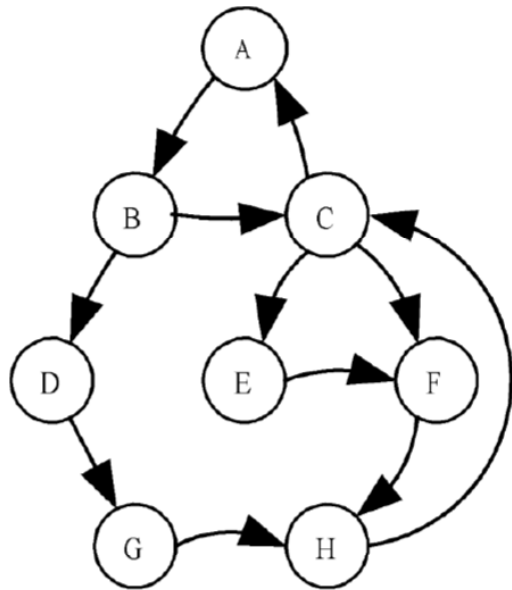
- 通过增加局部性，聚合是否减少了通信成本？
- 如果复制了计算，是否在一定范围的问题规模和处理器数量下，这种复制带来的好处超过了它的成本？
- 如果复制了数据，是否确认它不会限制问题规模或处理器数量，从而影响了算法的可伸缩性？
- 聚合产生的任务是否具有相似的计算和通信代价
- 任务数量是否仍然随着问题规模扩展？
- 任务数量是否适用于目标系统？
- 在聚合和代码修改成本之间权衡是否合理？



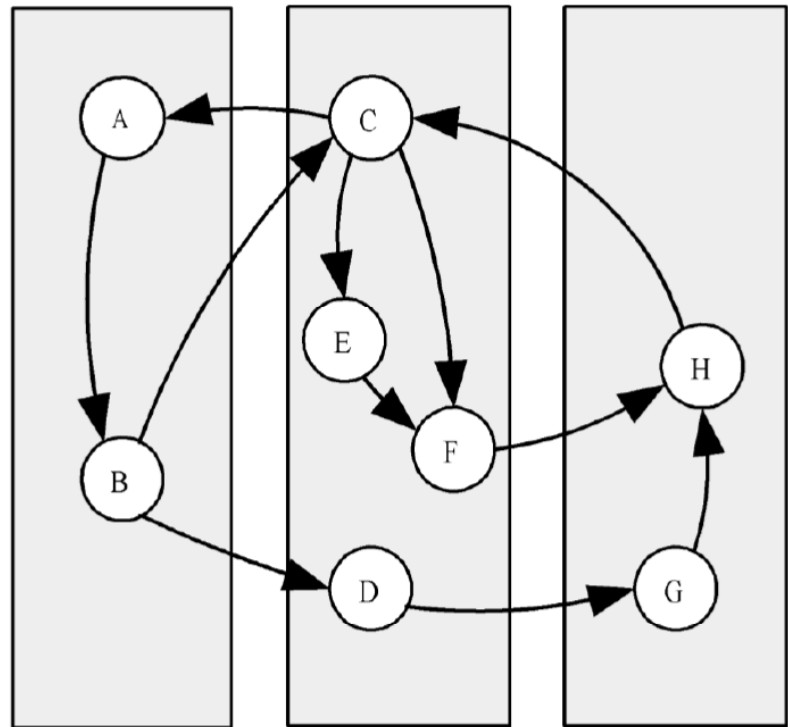
映射

- **阶段4. 映射**：将任务分配给处理器的过程。
- 开发映射算法的目标通常是**最小化总执行时间**。
映射的矛盾目标：
 - 最大化处理器利用率，即系统处理器在执行必要的问题解决任务时的平均使用时间百分比。（通过计算平衡）
 - 最小化处理器间通信（将通信进程映射到同一处理器）
- 为了最小化通信代价、并使每个进程/线程获得大致相同的工作量，**映射是必需的**。

映射示例



(a)



(b)



最优映射

- 寻找最优映射是NP难问题
- 必须依赖于启发式算法
- 例如...

问题清单：映射

- 考虑了基于每个处理器一个任务和多个任务的设计
- 评估了静态和动态任务分配
- 如果使用集中式负载平衡方案，是否已经验证管理器不会成为瓶颈？
- 如果使用动态负载平衡方案，是否已经评估了不同策略的相对成本？
- 如果使用概率或循环方法，是否有足够数量的任务以确保合理的负载平衡？



目录

- 算法设计方法论
- **案例研究**

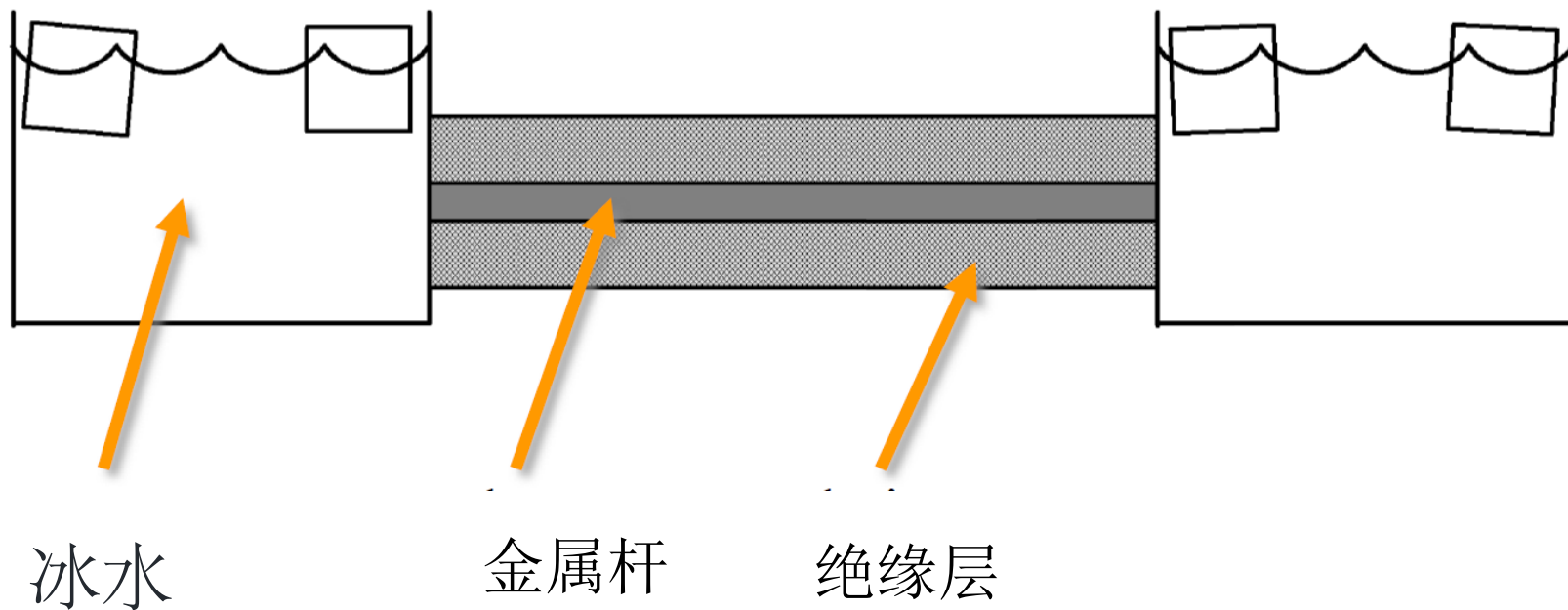


案例研究

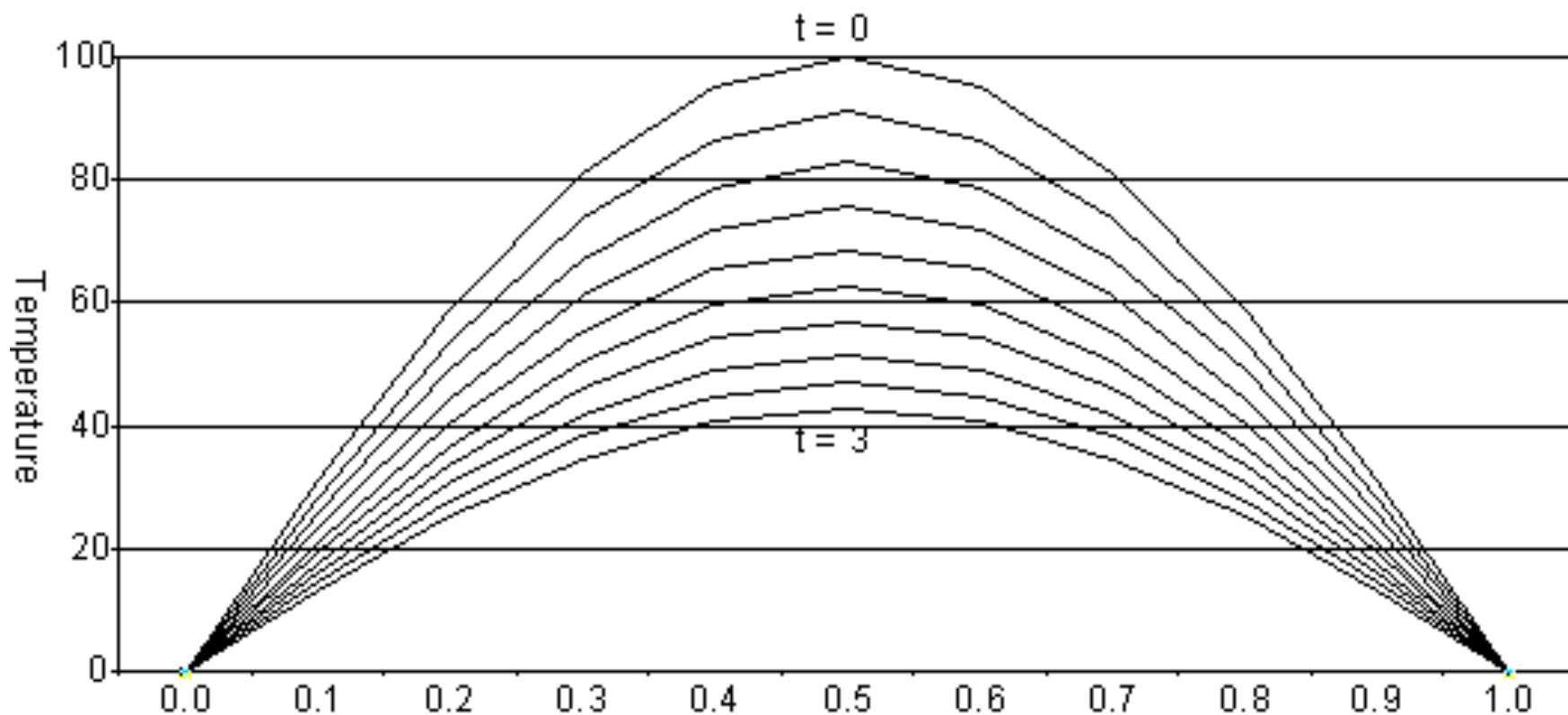
- 边界值问题
- 寻找极值
- n 体问题
- 添加数据输入



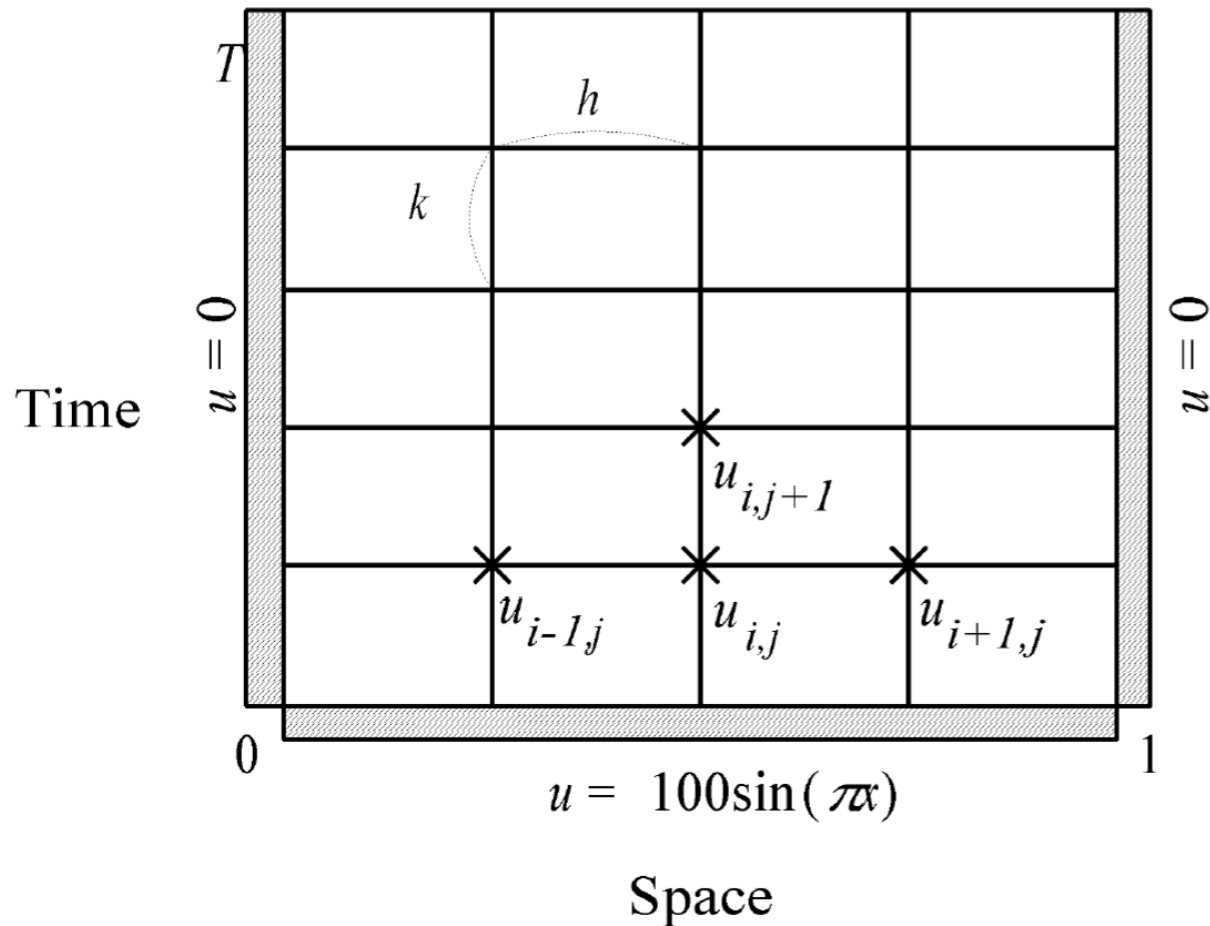
边界值问题



随着时间推移金属棒的温度



有限差分近似



$$u_{i,j} = ru_{i-1,j} + (1 - 2r)u_{i,j} + ru_{i+1,j}$$

$$r = k/h^2$$



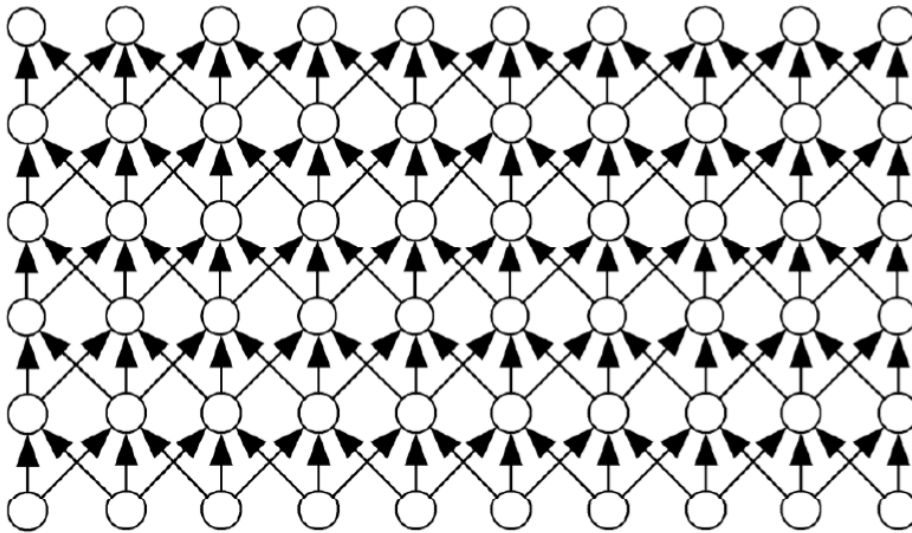
划分

- 每个网格点一个数据项
- 将每个网格点与一个基本任务关联
- 二维域分解

通信

- 识别基本任务之间的通信模式
- 每个内部基本任务都有三个进站通道和三个出站通道

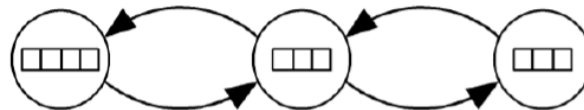
聚合和映射



(a)



(b)



(c)

聚合



串行执行时间

- χ — 更新元素的时间
- n — 元素（间隔）的数量
- m — 迭代次数
- 串行执行时间: $m (n-1) \chi$

并行执行时间

- p — 处理器数量
- λ — 消息延迟
- 并行执行时间 $m(\chi \lceil (n-1)/p \rceil + 2\lambda)$



对比 (n =间隔; m =时间)

| $n-1$ | m | 串行 | 任务/通道 当 $p \ll n-1$ |
|-------|-----|-----------------------|--|
| | | $m(n-1)\chi$ | $m(\chi \lceil (n-1)/p \rceil + 2\lambda)$ |
| 48 | 100 | 4800χ $p = 1$ | $600\chi + 200\lambda$ $p = 8$ |
| 48 | 100 | 同上 | $300\chi + 200\lambda$ $p = 16$ |
| 8K | 100 | $\sim(800K)\chi$ | $\sim 800\chi + 200\lambda$ $p = 1000$ |
| 64K | 100 | $\sim(6400K)\chi$ | $\sim 6400\chi + 200\lambda$ $p = 1000$ |

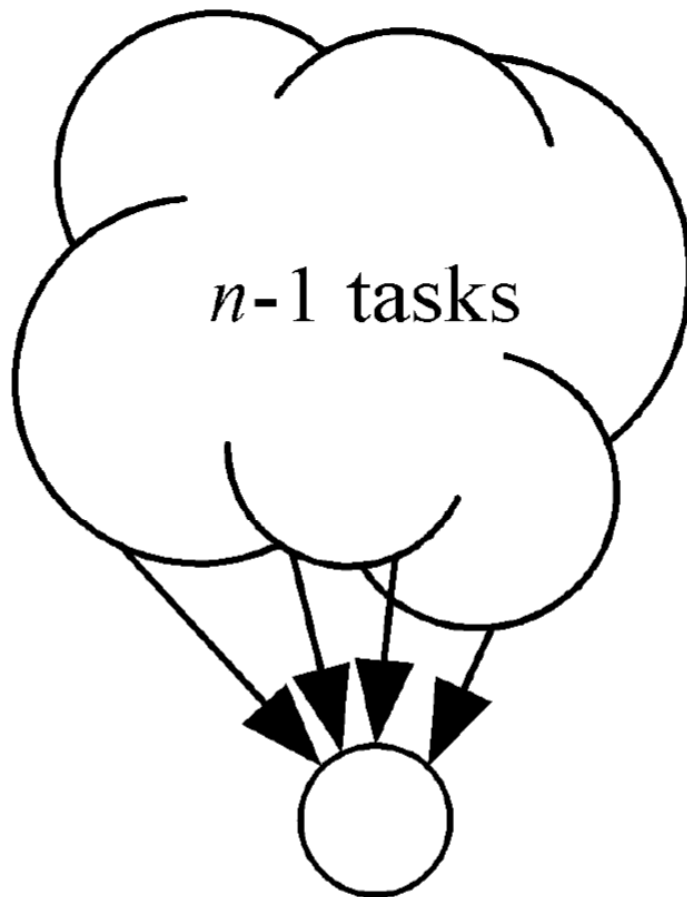


Reduciton(规约)

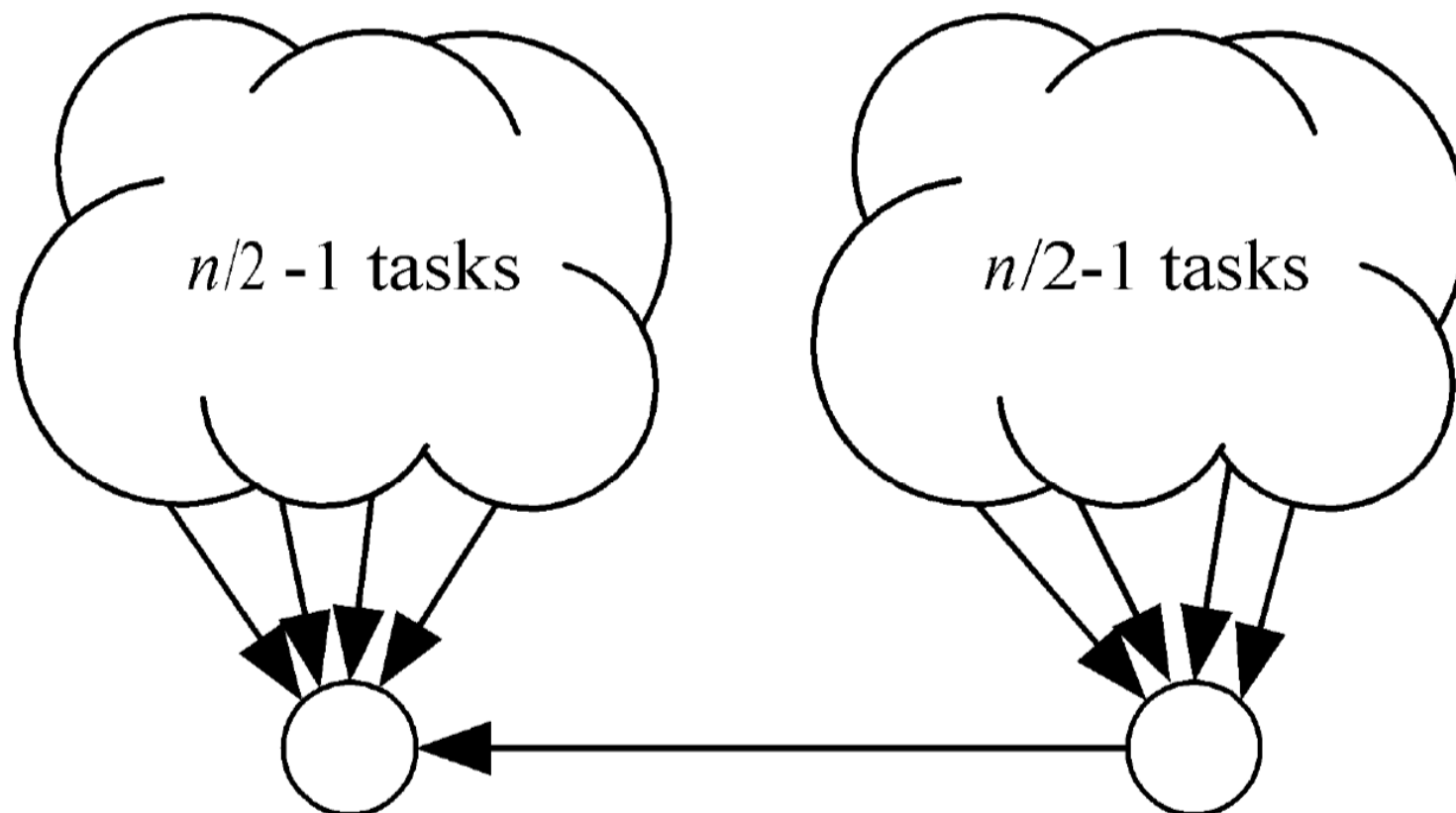
- 给定结合操作符： \oplus
- $a_0 \oplus a_1 \oplus a_2 \oplus \dots \oplus a_{n-1}$
- 例子
 - Add
 - Multiply
 - And, Or
 - Maximum, Minimum



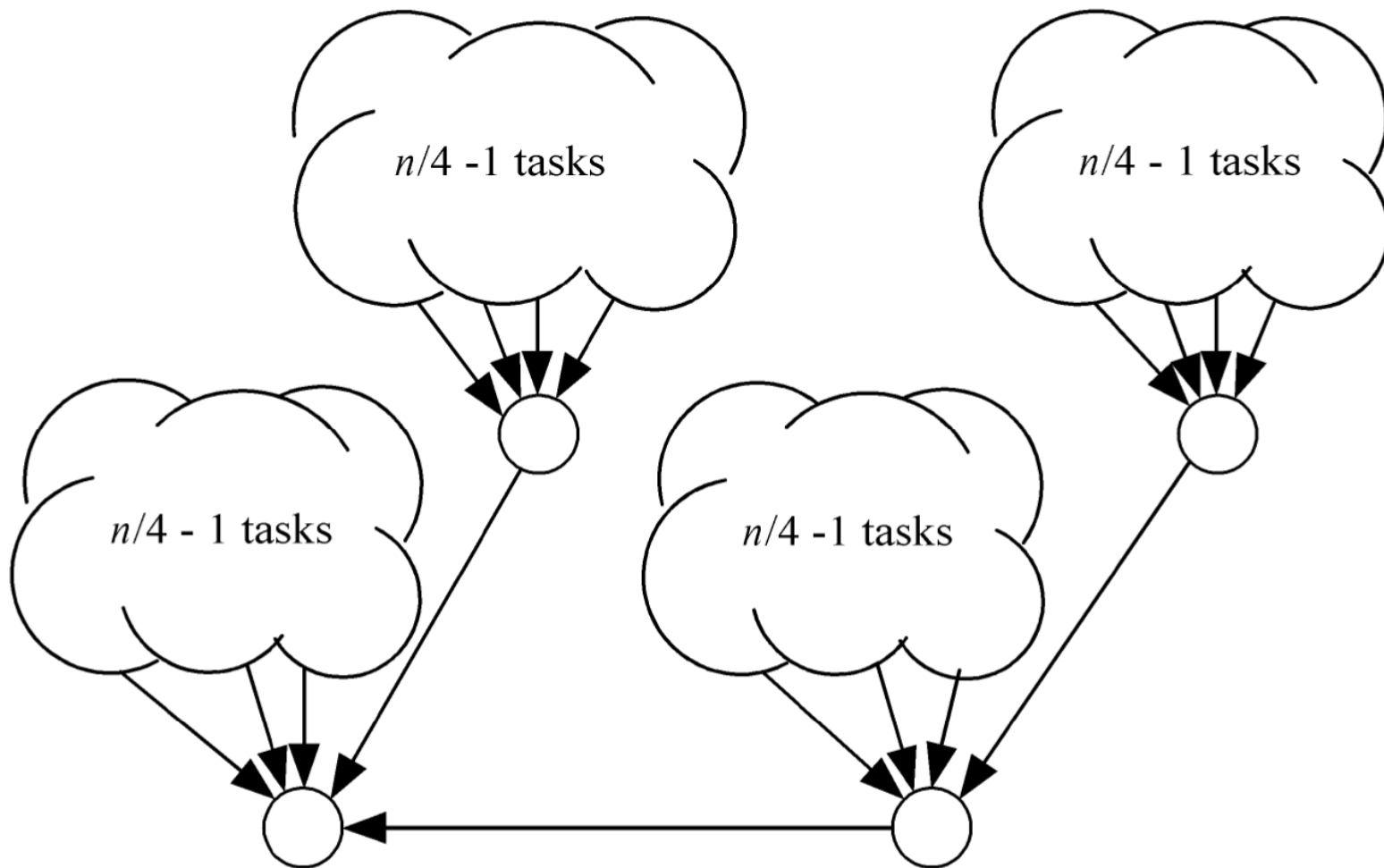
并行规约演化



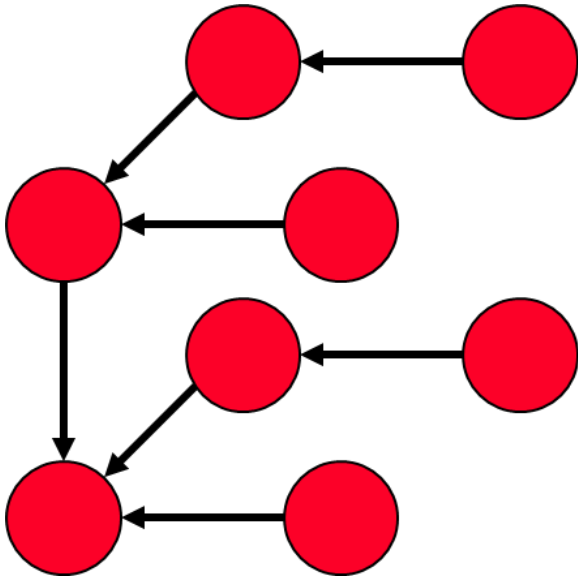
并行规约演化



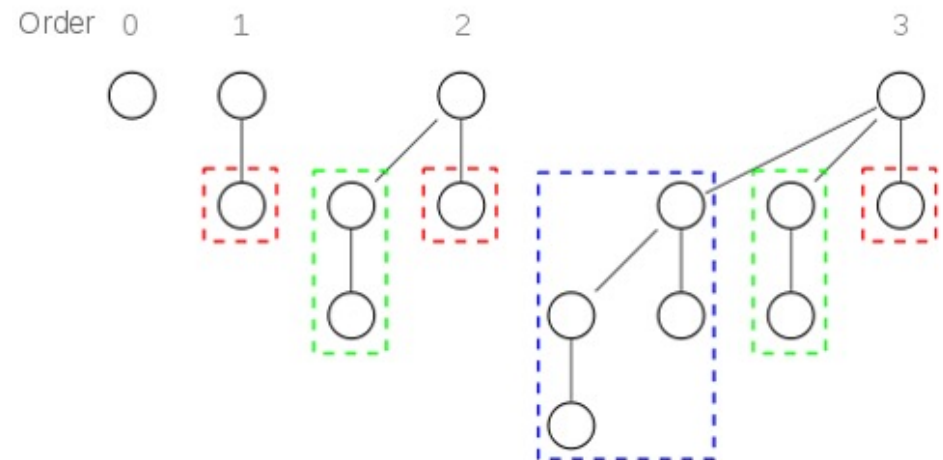
并行规约演化



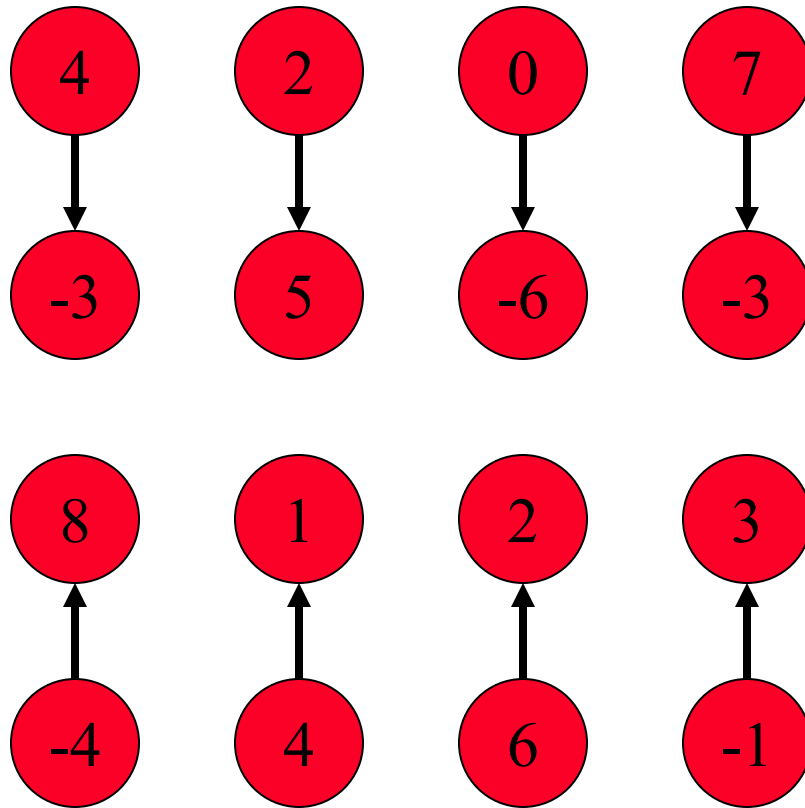
二项树



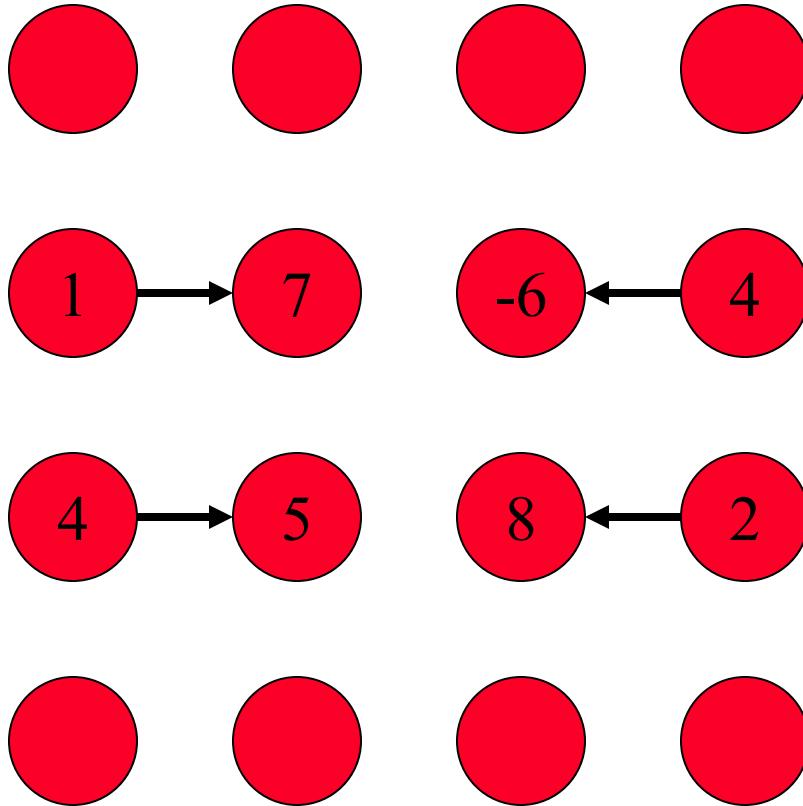
超立方体子图



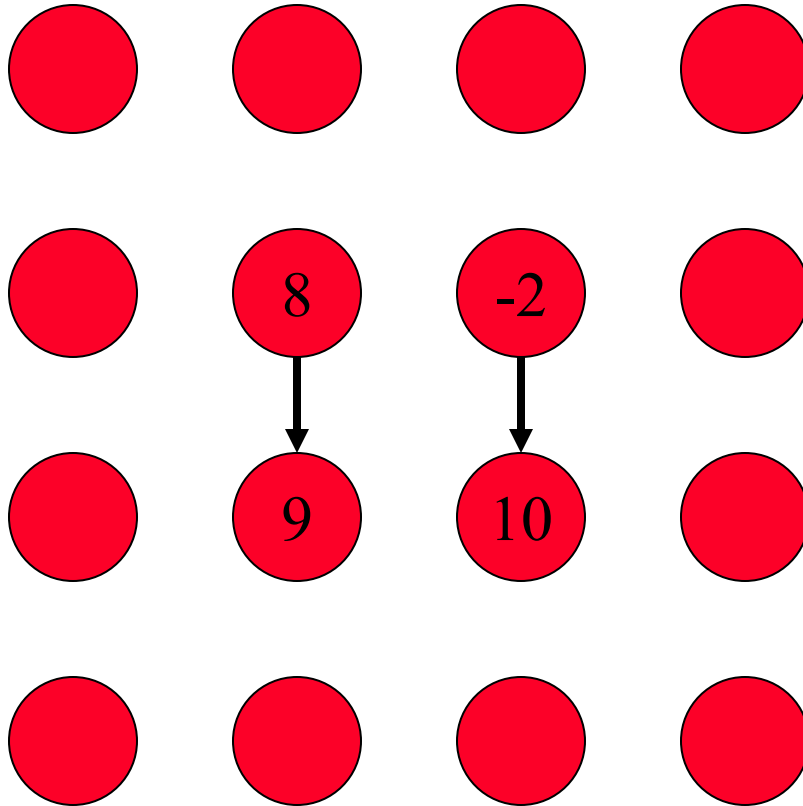
寻找全局和



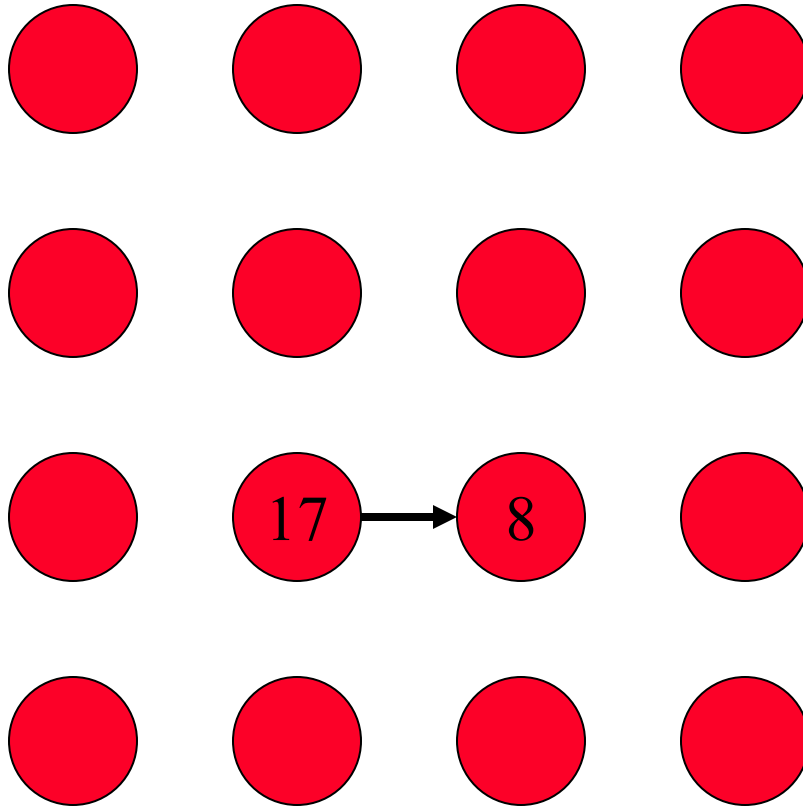
寻找全局和



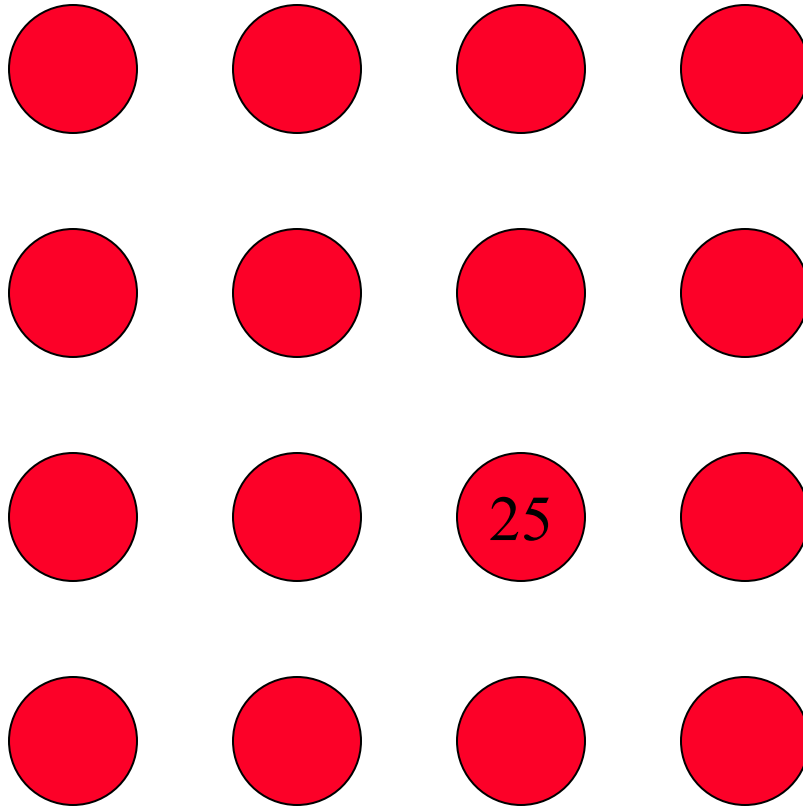
寻找全局和



寻找全局和



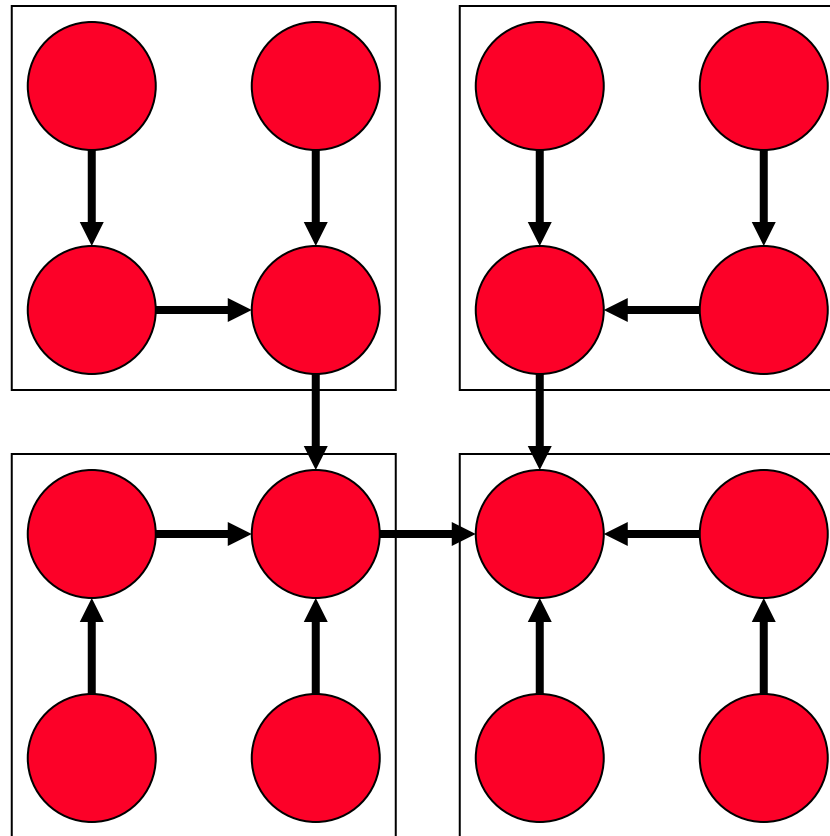
寻找全局和



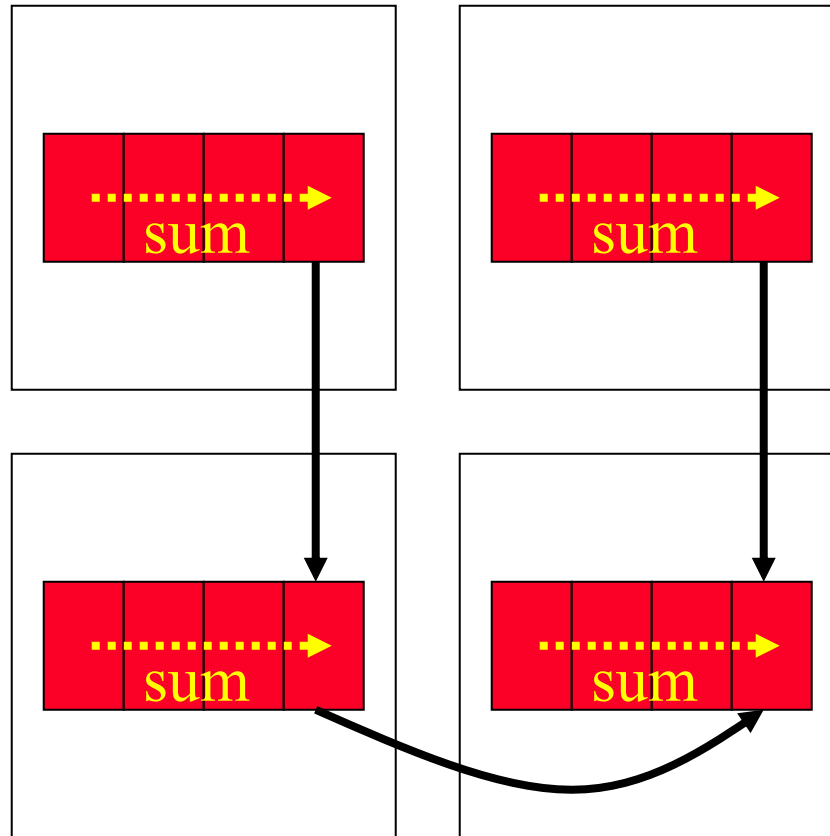
二项树



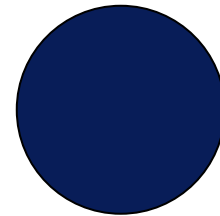
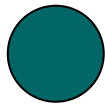
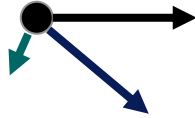
聚合



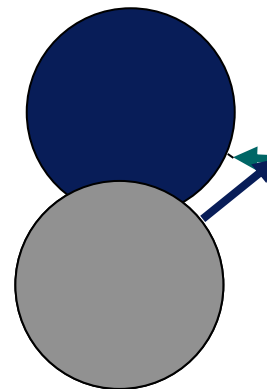
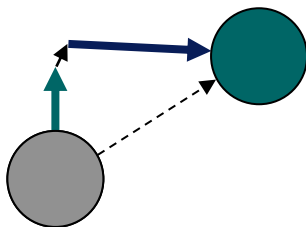
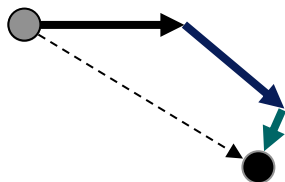
聚合



n体问题



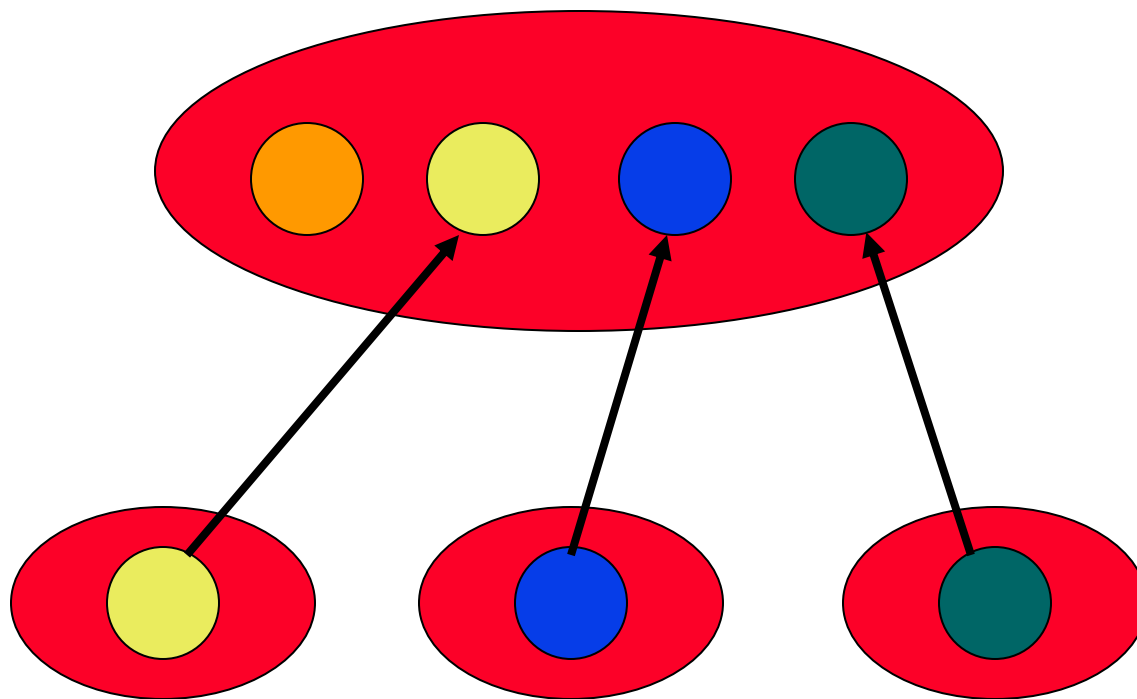
n体问题



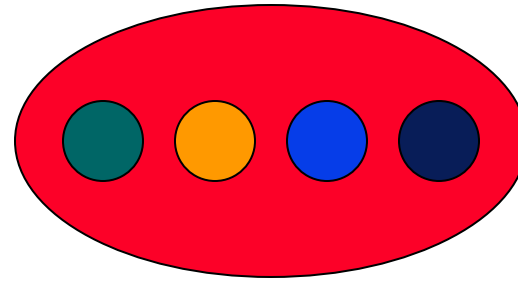
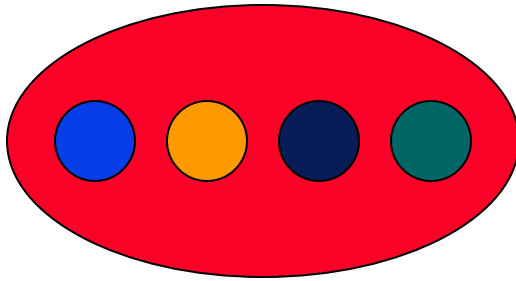
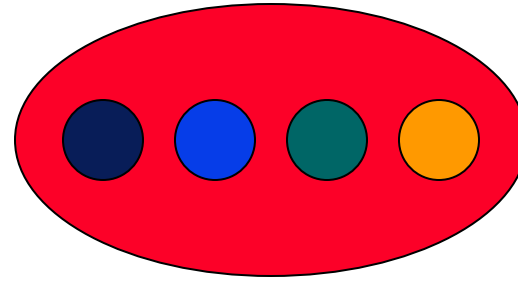
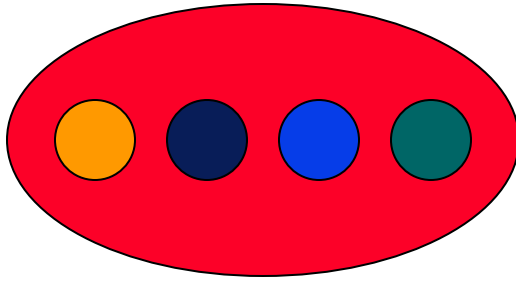
划分

- 域划分
- 假设每个粒子对应一个任务
- 任务具有粒子的位置和速度向量
- 迭代
 - 获取所有其他粒子的位置
 - 计算新的位置和速度

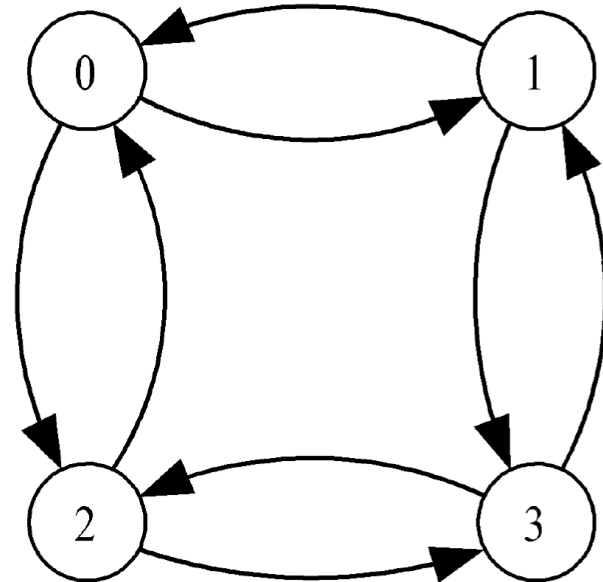
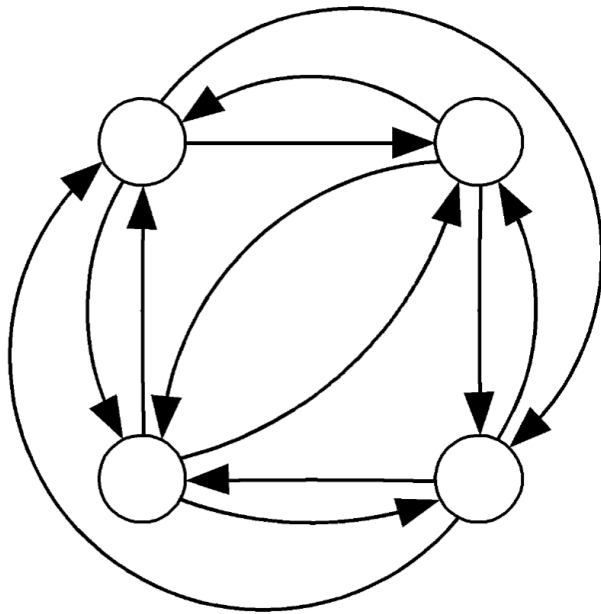
Gather



All-gather



完全图 vs. 超立方体



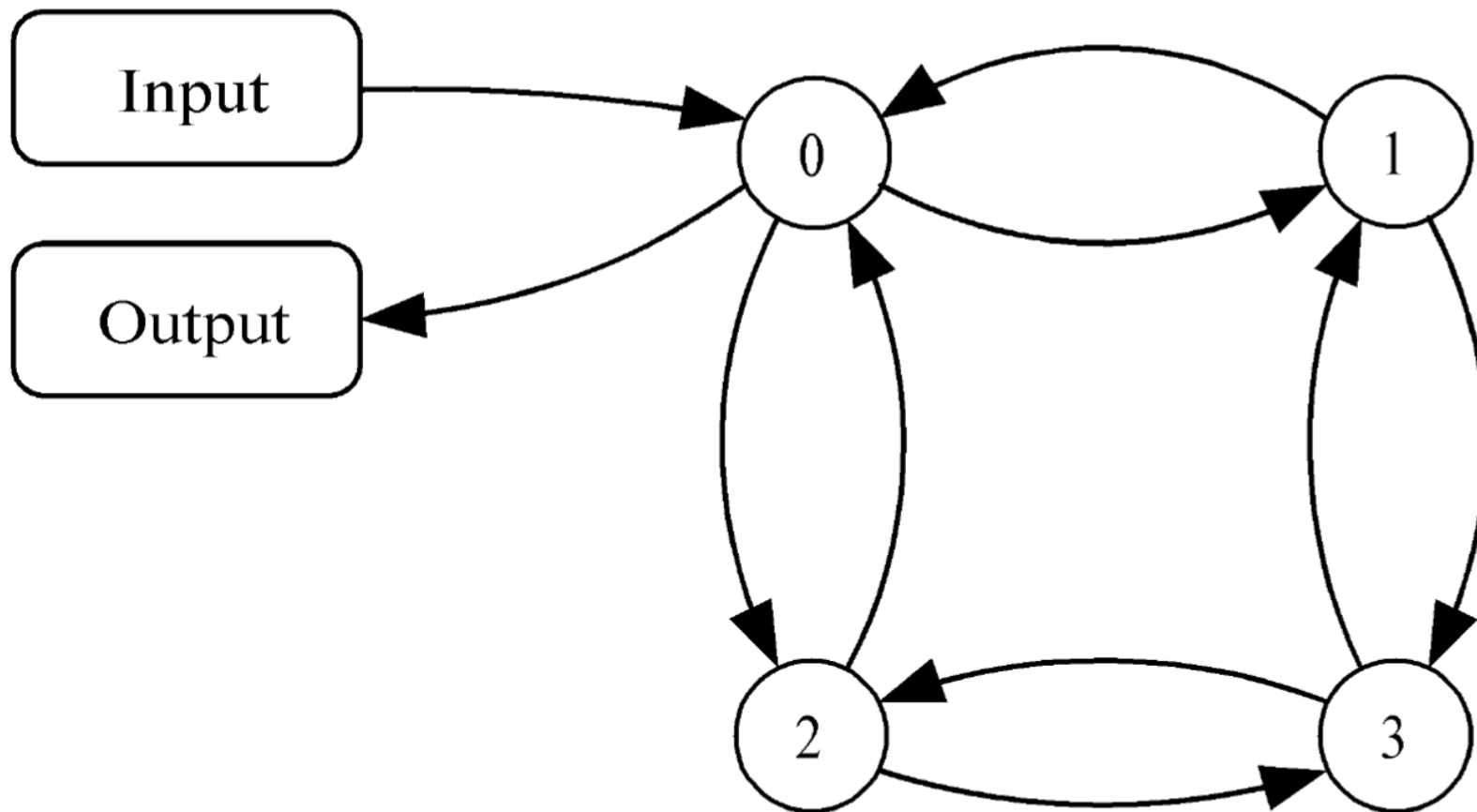
通信时间

完全图 $(p-1)(\lambda + \frac{n/p}{\beta}) = (p-1)\lambda + \frac{n(p-1)}{\beta p}$

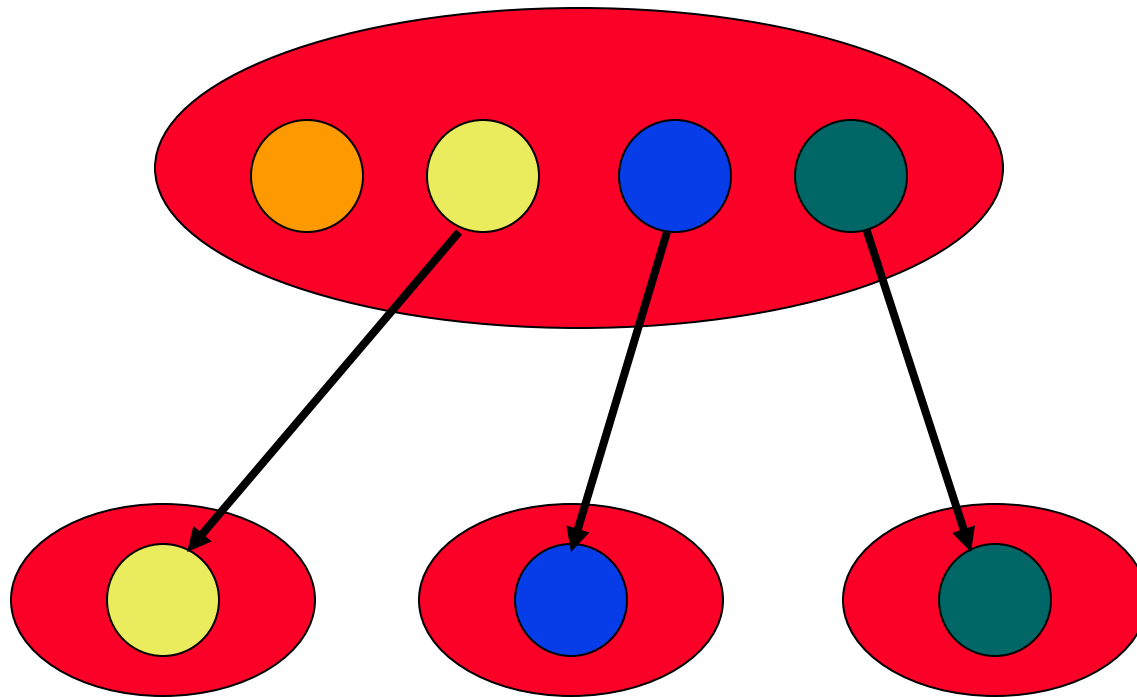
超立方体 $\sum_{i=1}^{\log p} \left(\lambda + \frac{2^{i-1}n}{\beta p} \right) = \lambda \log p + \frac{n(p-1)}{\beta p}$



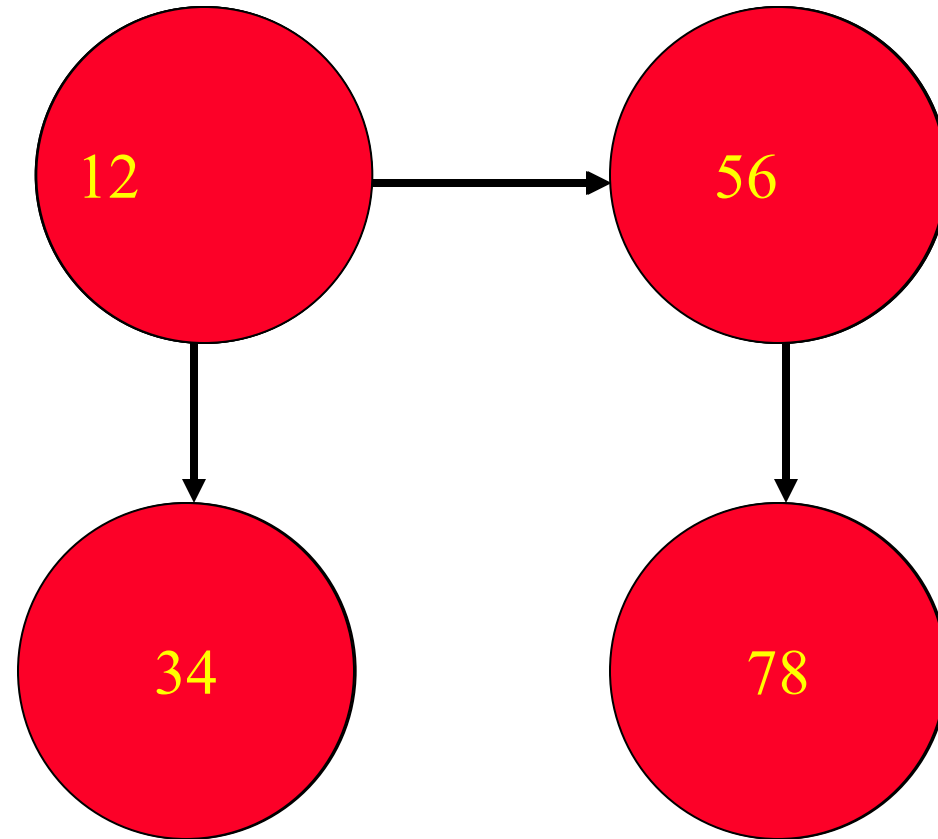
添加数据输入



Scatter



Scatter in $\log p$ Steps



总结：设计步骤

- 划分计算
- 聚合任务
- 将任务映射到处理器
- 目标
 - 最大化处理器利用率
 - 最小化处理器间通信



总结：基础算法

- Reduciton(规约)
- Gather 和 Scatter
- All-gather