

# 并行程序设计模型

---

## ■ 并行程序设计模型

- 隐式并行 (Implicit Parallel)
- 数据并行 (Data Parallel)
- 共享变量 (Shared Variable)
- 消息传递 (Message Passing)

# 隐式并行

---

## ■ 隐式并行

- 编写串行程序
- 通过编译器和运行支持系统将串行程序自动并行化
- 特点：语义简单，可移植性好，易于调试和验证
- 缺点：细粒度并行，效率很低

# 数据并行

---

## ■ 数据并行

- SIMD（单指令流多数据流）
- 同一操作同时作用到一组数据上
- 特点：单线程，单一地址空间，编程简单，松散同步，隐式交互，隐式数据分配
- 缺点：并行粒度局限于数据级并行，粒度小
- 典型代表：Fortran 90，HPF

**SIMD: Single Instruction Multiple Data**

# 共享变量

---

## ■ 共享变量

- 适用于 SMP 和 DSM
- 特点：松散同步，多线程（SPMD，MPMD）  
单一地址空间，显式同步，隐式通信，隐式数据分布
- 典型代表：OpenMP，Pthreads

**SMP: Shared Memory Processors**

**DSM: Distributed Shared Memory**

**SPMD: Single Program Multiple Data**

**MPMD: Multiple Program Multiple data**

**OpenMP: Open Multi-Processing**

**Pthreads: POSIX threads**

# 消息传递

---

## ■ 消息传递

- MPP、COW 的自然模型
- 特点：异步并行，多线程，多地址空间，  
显式同步，显式通信，显式数据映射和负载分配
- 典型代表：MPI, PVM

**MPP: Massively parallel processing**

**COW: Cluster of Workstations**

**MPI: Message Passing Interface**

**PVM: Parallel Virtual Machine**

# 并行编程模型

---

## ■ 并行编程模型标准

- 数据并行: Fortran 90, HPF; 适用于 SMP, DSM
- 共享内存: OpenMP, Pthreads; 适用于 SMP, DSM
- 消息传递: MPI, PVM; 适用于所有并行机
- 三者可混合使用

# 并行化方法

---

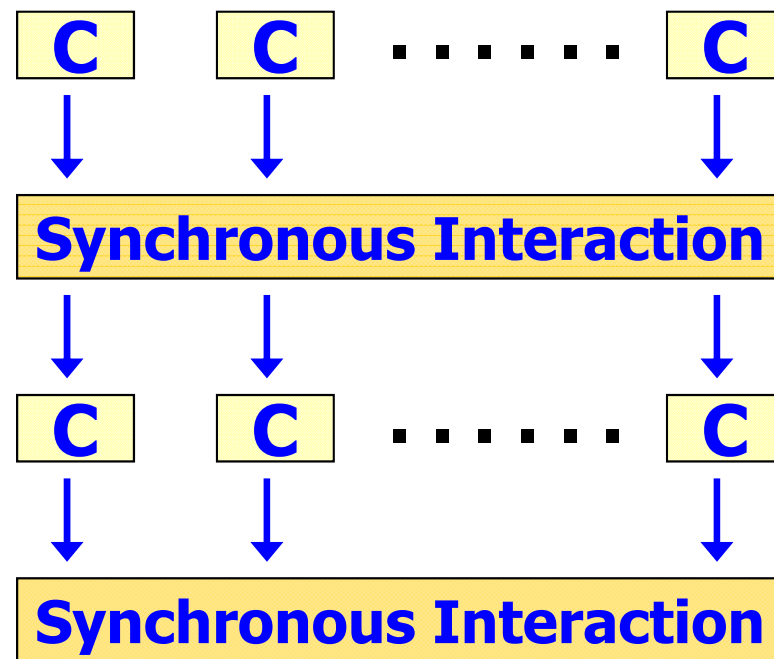
## ■ 基本并行化方法

- 相并行 (Phase Parallel)
- 流水线并行 (Pipeline Parallel)
- 主从并行 (Master-Slave Parallel)
- 分而治之并行 (Divide and Conquer Parallel)
- 工作池并行 (Work Pool Parallel)

# 并行化方法

## ● 相并行

- 一组超级步（相）
- 步内各自计算
- 步间通信同步
- 方便差错和性能分析
- 计算和通信不能重叠



---

## ● 流水线并行

- 将分成一系列子任务  $t_1, t_2, \dots, t_m$ ，一旦  $t_1$  完成，后继的子任务就立即开始，并以同样的速率进行计算
- 一组进程，流水线作业，流水线设计技术

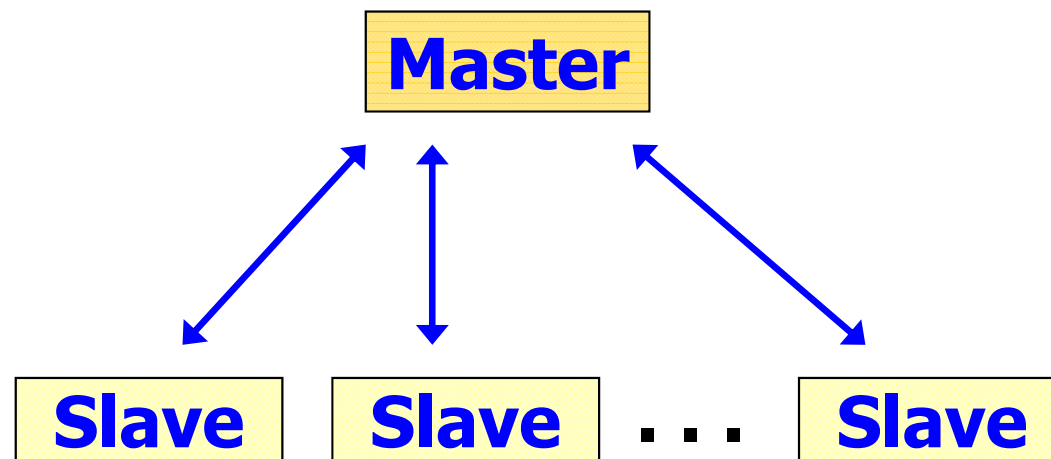




# 并行化方法

- 主从并行

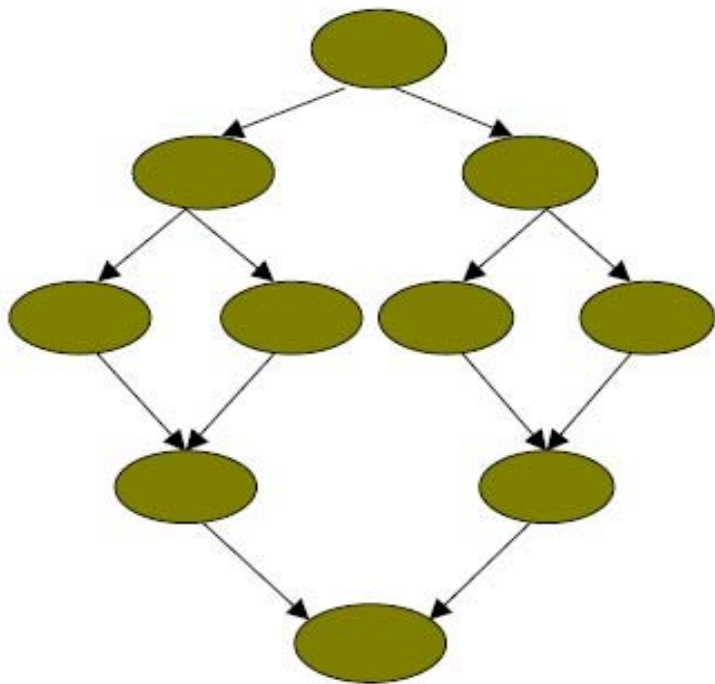
- 主进程：串行，协调任务
- 子进程：计算子任务
- 与相并行结合
- 主进程易成为瓶颈



---

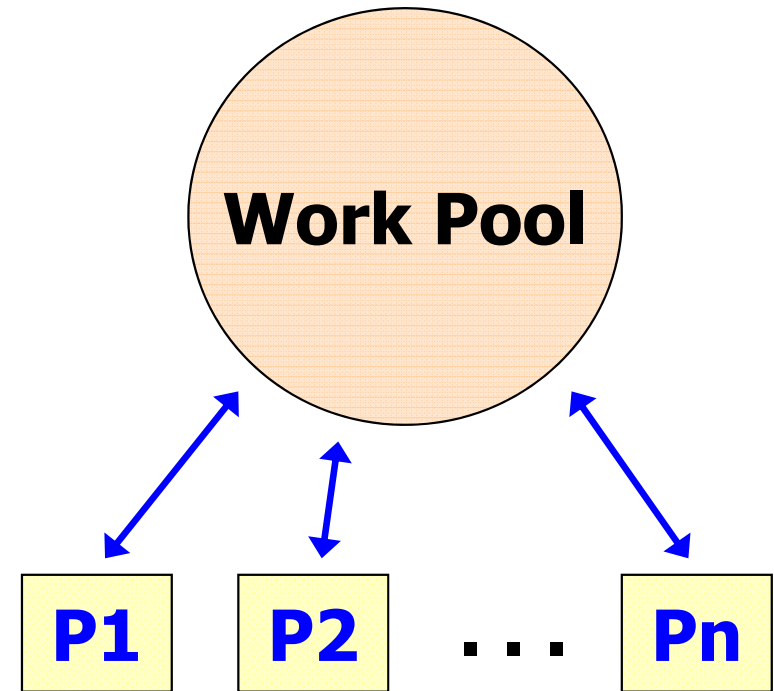
- 分而治之

- 将问题分解成若干特征相同的子问题，分而治之
- 父进程把负载分割并指派给子进程
- 重点在于归并
- 难以负载平衡



# 并行化方法

- 工作池并行
  - 初始状态：一件工作
  - 进程从池中取任务执行
  - 可产生新任务放回池中
  - 直至任务池为空
  - 易于负载均衡



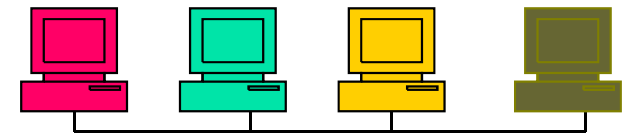
# 分而治之

---

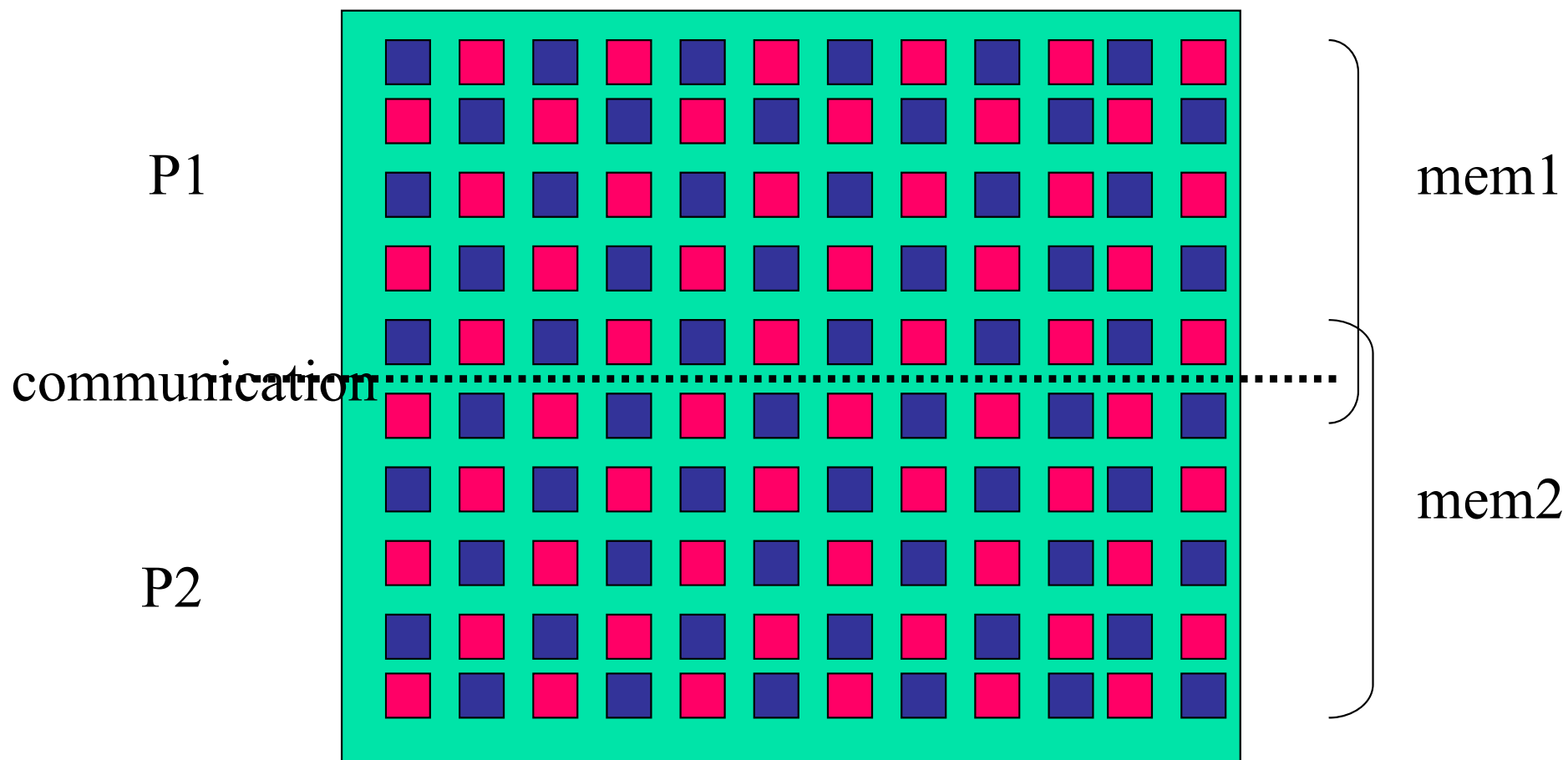
$$Y=(A+B(C+DEF))+G \quad 6$$

$$Y=(\underline{A+G})+\underline{B(C+DEF)} \quad 5$$

$$Y=(\underline{A+G+BC})+\underline{BD^*EF} \quad 3$$

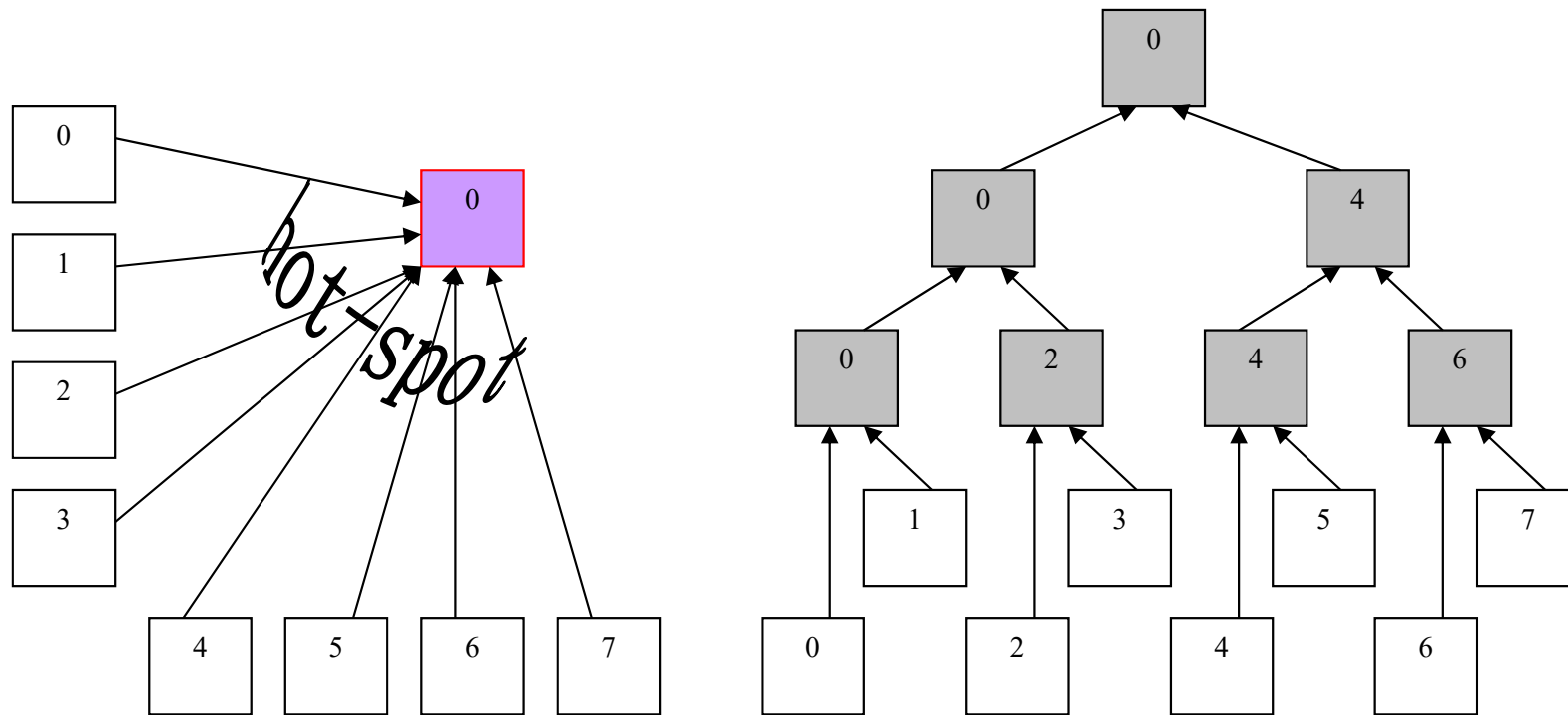


# 红黑格法



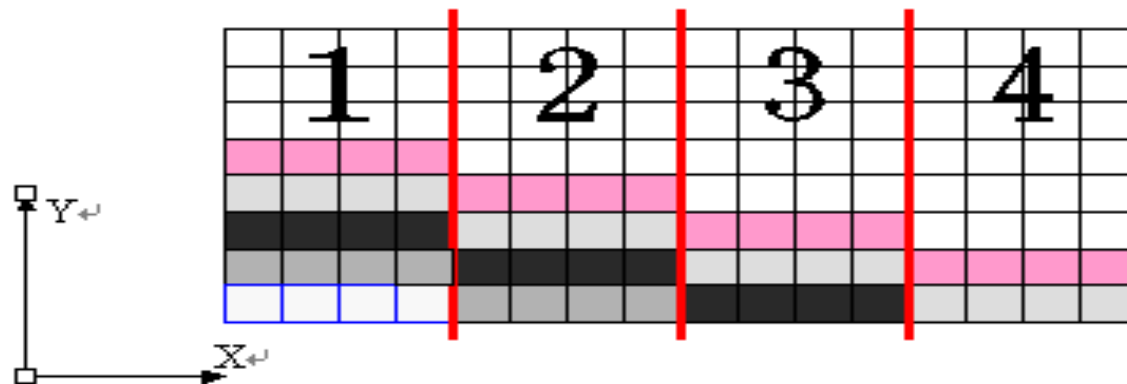
# 并行化方法






- 并行二叉树技术：解决通信瓶颈问题，通信复杂度 $O(P)$ 下降到 $O(\log P)$



# 并行化方法

- **并行流水线技术：** 解决串行算法中不可避免的数据相关性，例如Gauss-Seidel迭代、双曲型方程中的上下游流场数据依赖问题等



1. P1 计算线 ( $I=1-4, J=1$ ), P2-P4 空闲; 
2. P1 计算线 ( $I=1-4, J=2$ ), P2 计算线 ( $I=1-4, J=1$ ), P3-P4 空闲; 
3. P1 计算线 ( $I=1-4, J=3$ ), P2 计算线 ( $I=1-4, J=2$ ), P3 计算线 ( $I=1-4, J=1$ ), P4 空闲; 
4. P1 计算线 ( $I=1-4, J=4$ ), P2 计算线 ( $I=1-4, J=3$ ), P3 计算线 ( $I=1-4, J=2$ ), P4 计算线 ( $I=1-4, J=1$ ); 
5. P1 计算线 ( $I=1-4, J=5$ ), P2 计算线 ( $I=1-4, J=4$ ), P3 计算线 ( $I=1-4, J=3$ ), P4 计算线 ( $I=1-4, J=2$ ); 
6. 依次类推, 可并行计算, 只要 Y 方向网格点数足够多; ↩

# 如何做并行应用？

---

- 串行算法分析：模块化
- 串行软件测试：找消耗90%CPU的hot
- 选择并行方法：分而治之——从物理模型开始
- 并行软件框架设计：流水线/主从/平等
- 并行程序设计：MPI/PVM/OPEN MP等
- 性能测试与分析：正确性、效率
- 进一步优化：应用

# 并行算法设计原则

## ■ 并行算法设计基本原则 Parallel Programming Paradigms

- 与体系结构相结合
- 具有可扩展性
- 粗粒度
- 减少通信
- 优化性能

**粒度**是指各个线程可以独立并行执行的任务的大小，是一个相对的概念，与并行度和并行机相关。一般可理解为：

- 细粒度：基于向量和循环级并行
- 中粒度：较大的循环级并行
- 大粒度：任务级并行（如：区域分解）



# 并行程序设计步骤

---

## ■ 并行程序设计步骤

- 划分（Partitioning）

将计算任务划分成尽可能多的小任务

划分方法主要有：数据分解（区域分解）和功能分解

- 通信（Communication）

确认各任务间的数据交流，评估任务划分的合理性

- 组合（Agglomeration）

依据任务的局部性，将小任务组合成大任务，减少通信

- 映射（Mapping）

将组合后的任务分配到各个线程，力争负载平衡