1.名词解释：

UMA（Uniform Memory Access）：均匀访存模型（12，54）

NUMA（Nonuniform Memory Access）：非均匀存储访问模型（12，54）

COMA（Cache-Only Memory Access）：全高速缓存存储访问（13，54）

NORMA（No-Remove Memory Access）:非远程存储访问模型（15）

SPMD（Single Program Multiple Data）:单程序多数据

SIMD（Single Instruction Multiple Data）:单指令多数据

MIMD（Multiple Instruction Multiple Data）:多指令多数据流

UMA（Uniform Memory Access）：所有处理器对内存的访问时间是相同的，每个处理器可以直接访问相同的物理内存，因此系统中所有处理器的存储器访问速度相同。

NUMA（Nonuniform Memory Access）：每个处理器拥有附属的部分共享存储器，不同的处理器访问不同的内存区域的速度可能会不同。

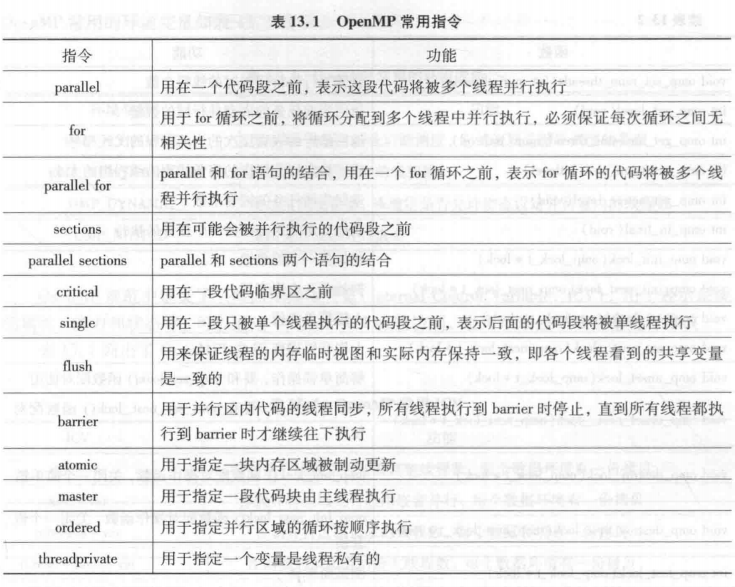
COMA（Cache-Only Memory Access）：每个处理器拥有附属的部分共享存储器，这些共享存储器由高速cache组成。通过将主存储器中的所有数据都缓存到高速缓存中，并且让所有处理器都可以直接访问高速缓存，从而实现了对内存访问的高效处理。

SPMD（Single Program Multiple Data）：SPMD是单程序多数据流并行计算的编程模型，其中多个处理器执行同一个程序，但是每个处理器处理的数据是不同的。在SPMD模型中，程序的控制流是相同的，但是每个处理器的数据流可以不同。

SIMD（Single Instruction Multiple Data）：SIMD指的是单指令多数据流并行计算的编程模型。在SIMD模型中，多个处理器可以执行相同的指令，但是每个处理器处理的数据是不同的。

MIMD（Multiple Instruction Multiple Data）：MIMD指的是多指令多数据流并行计算的编程模型。在MIMD模型中，多个处理器可以执行不同的指令，且每个处理器处理的数据也可以不同。这种模型的并行计算可以支持更加复杂的任务，并且可以实现更高的计算性能和效率。

OpenMP相关指令含义



parallel指令(P282)：产生多个线程，即创建一个并行域

for指令(P285)：将一个for循环分配到多个线程中执行

section指令(P291)：用在section结构里，用来将sections结构里的代码划分成几个不同的段，每段都并行执行

simd指令(P294)：应用于循环，指示循环可以转换成simd循环，可以使用simd指令并发执行循环的多次迭代

single指令(P295)：标识区域仅由一个线程执行。

private子句(P296)：将一个或多个变量声明成线程私有的变量，变量声明成私有变量后，指定每个线程都有它自己的变量私有副本。

firstprivate子句(P297)：指定每个线程都有它自己的变量私有副本，并且变量要被继承主线程中的初值。

lastprivate子句(P298)：实现在退出并行区域时将私有变量的值赋给共享变量

threadprivate指令(P299)：指定全局的对象被各个线程各自复制了一个私有的拷贝，即各个线程具有各自私有的全局对象。

shared子句(P300)：用来声明一个或多个变量是共享变量。

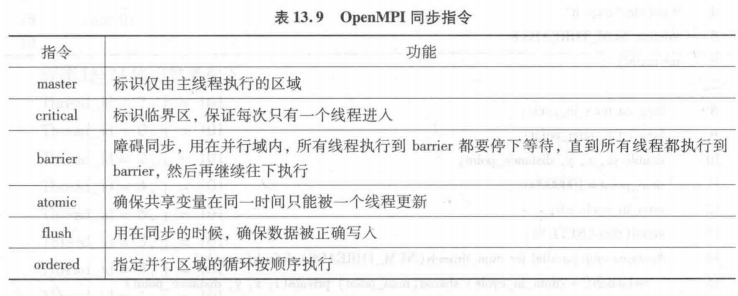
reduction子句(P302)：指定一个或多个变量是私有的，并且在并行处理结束后对这些变量执行指定的归约操作，然后将结果返回给主线程中的同名变量。

if子句(P304)：指定编译制导满足的条件

copyin子句(P304)：将主线程中threadprivate变量的值拷贝到执行并行区域的各个线程的threadprivate变量中，从而使得组内的子线程都拥有和主线程同样的初始值。

copyprivate子句(P305)：提供了用一个私有变量将一个值从一个线程广播到执行同一并行区域的其他线程的机制。

default子句(P306)：允许用户控制并行区域中变量的共享属性。



(P308-P315)

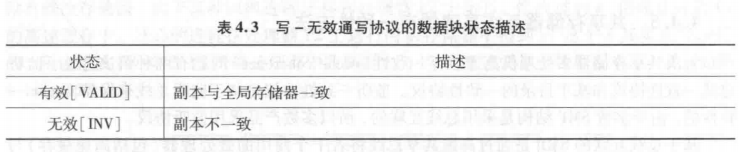
1. 高速缓存不一致的原因(P61)：

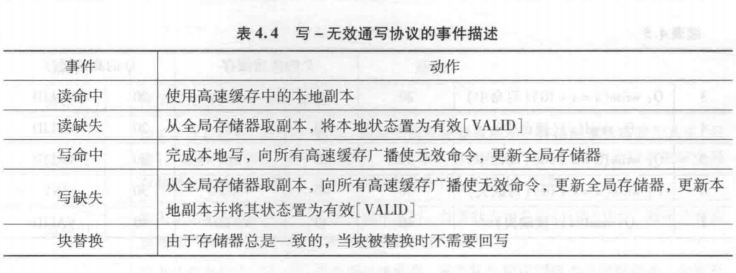
高速缓存与存储器不一致：

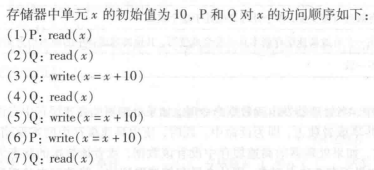
1. 处理器写高速缓存，由于采用回写策略，没有立即写存储器
2. I/O处理机或I/O设备绕过处理器写存储器

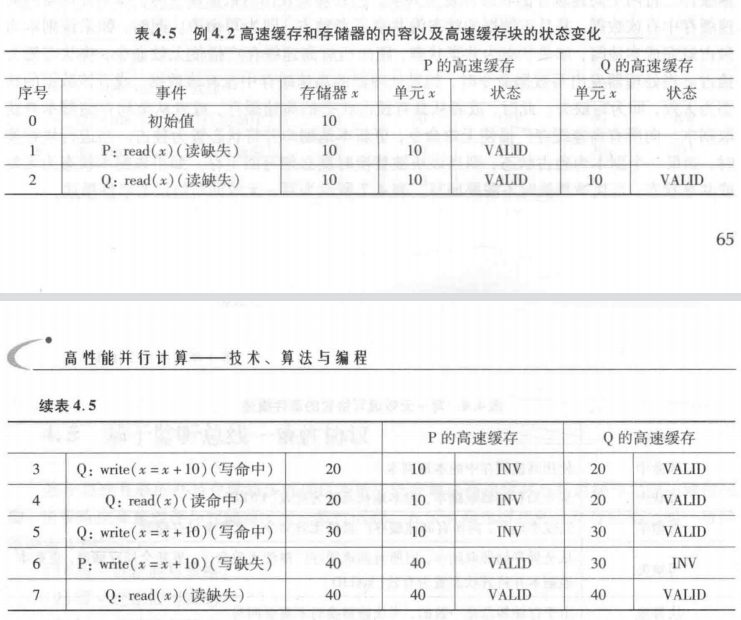
高速缓存与高速缓存不一致的原因：

1. 共享数据写操作引起高速缓存不一致
2. 进程迁移引起高速缓存不一致
3. 绕过高速缓存的I/O操作引起高速缓存不一致
4. 高速缓存一致性协议
5. 写-无效通写协议

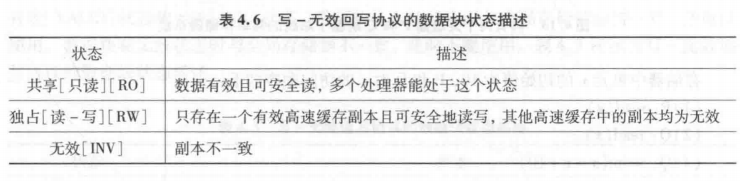


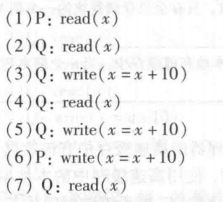






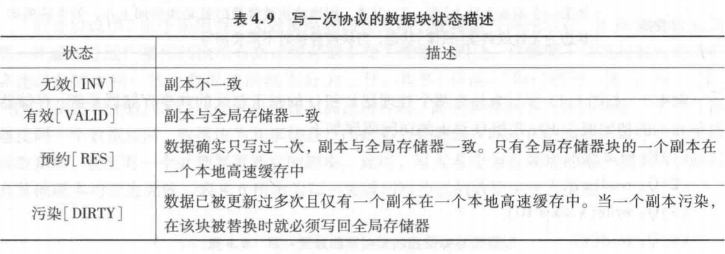
1. 写-无效回写协议

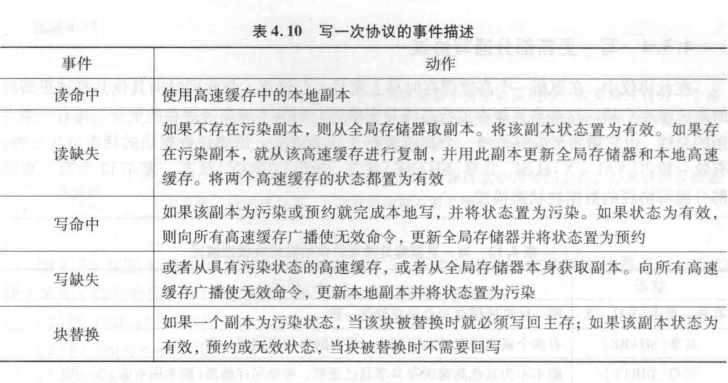


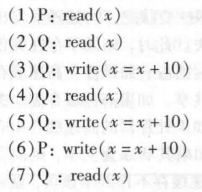




1. 写一次协议

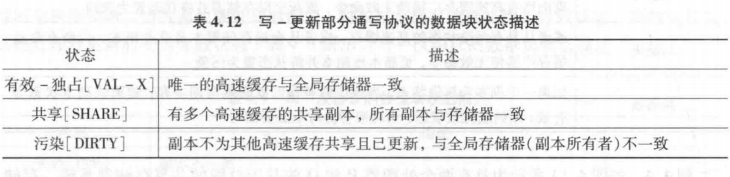


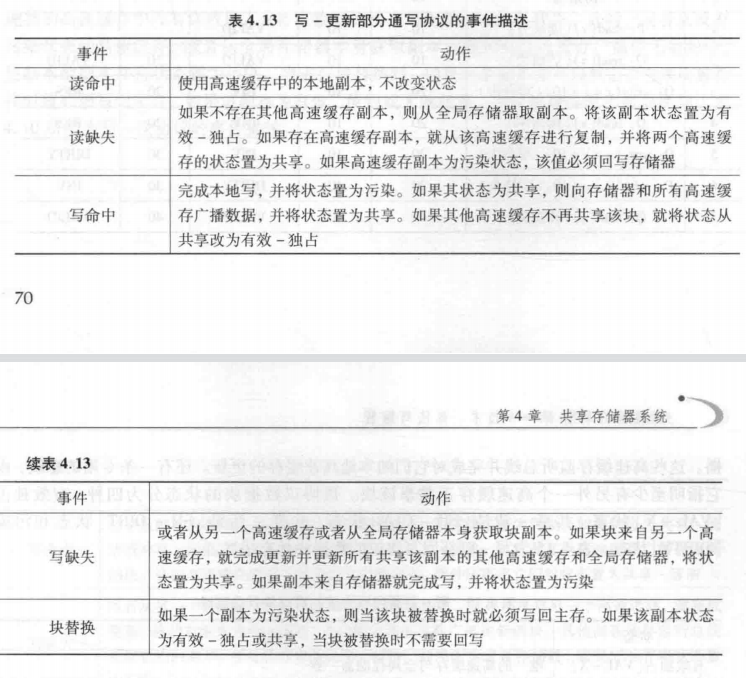


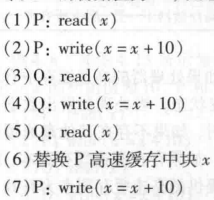




1. 写-更新部分通写协议

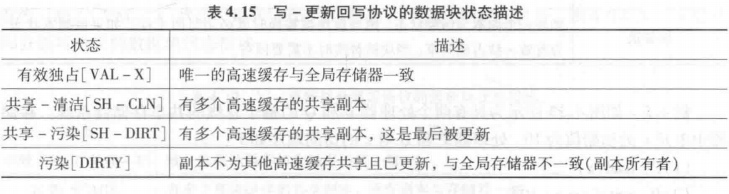


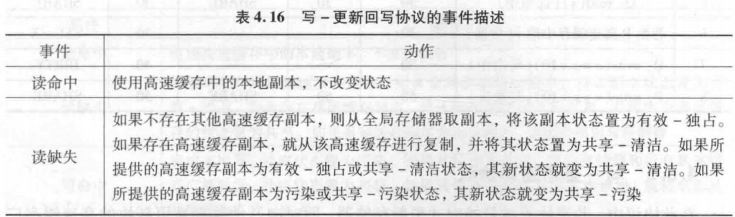


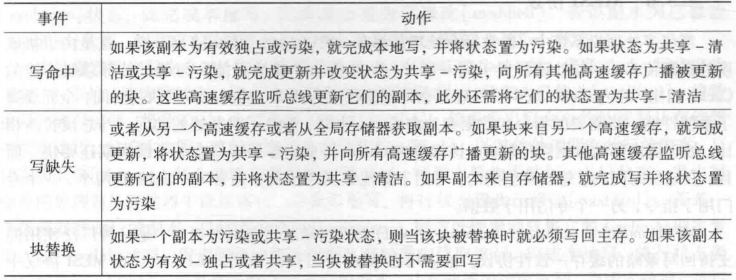


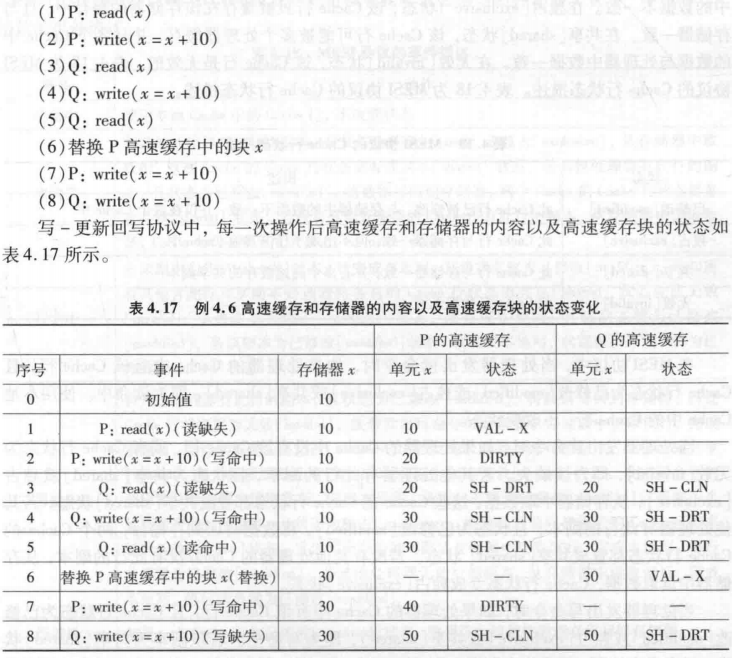


1. 写-更新回写协议

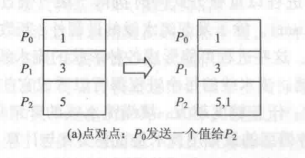




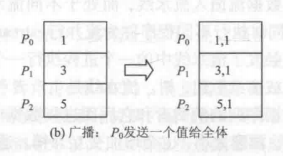


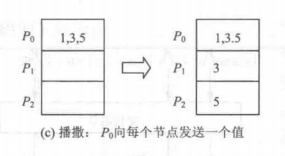


1. **进程间交互方式(P183)：**
2. 一对一：点到点(point to point)

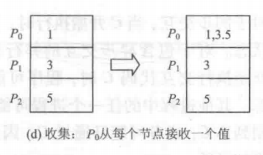


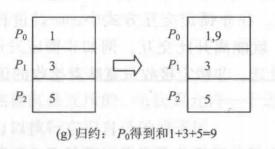
1. 一对多：广播(broadcast)和播撒(scatter)



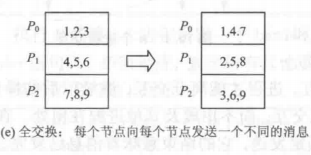


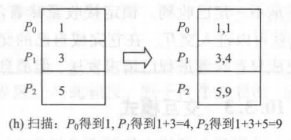
1. 多对一：收集(gather)和归约(reduce)

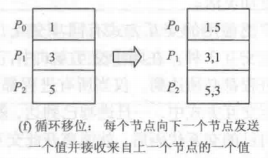


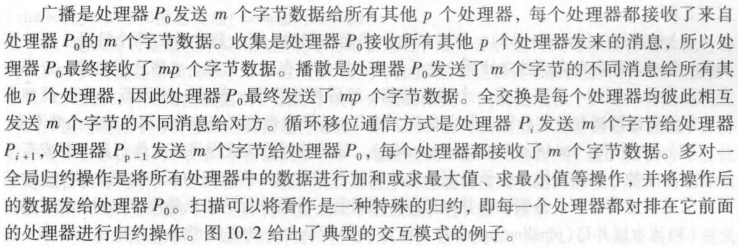


1. 多对多：全交换(total exchange)和循环移位(circular shift)









1. 指令级并行(P115)

**指令级并行：**指令级并行是计算机中提高指令执行效率的一种技术。当指令之间不存在相关时，它们可以在流水线中重叠并行执行，这样就可以提高CPU的吞吐量。指令级并行包括多种技术，如流水线、超标量技术、动态执行调度等。

**乱序执行：**CPU允许将多条指令不按程序规定的顺序分开发送给各个相应的电路处理单元的技术。

**寄存器重命名技术：**当一条指令写一个结果寄存器时不直接写到这个结果寄存器，而是先写到一个中间寄存器过渡一下，当这条指令提交的时候再写到结果寄存器中**。**有两个作用：一是消除指令之间的寄存器读后写相关(write after read,WAR)，和写后写相关(write after read,WAW)；二是当指令执行发生例外或转移指令猜测错误而取消后面的指令时可以保证现场的精确。

**分支预测技术：**是为了解决处理器处理分支指令时遇到的由于判断条件的真/假的不同，有可能会产生跳转，这会打断流水线中指令的处理。分支预测技术包含编译时进行的静态分支预测和硬件在执行时进行的动态分支预测。最简单的静态分支预测是任选一条分支，最简单的动态分支预测是分支预测缓冲区或分支历史表。

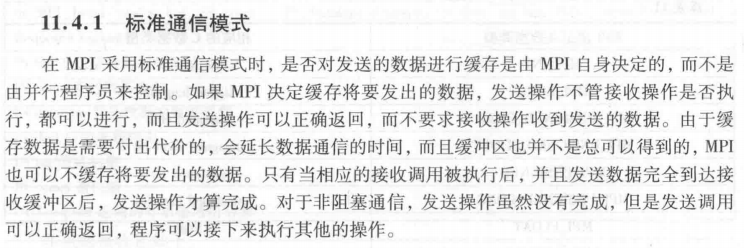
1. **MPI通信模式 (199)**

四种通信模式：标准通信模式、缓存通信模式、同步通信模式和就绪通信模式。

* 标准通信模式（Standard Mode）：在标准通信模式中，发送进程通过MPI\_Send函数向接收进程发送消息，并由MPI\_Recv函数接收消息。在标准通信模式中，发送进程会一直阻塞，直到接收进程成功接收到消息。
* 缓存通信模式（Buffered Mode）：在缓存通信模式中，发送进程通过MPI\_Bsend函数向接收进程发送消息，并由MPI\_Recv函数接收消息。在缓存通信模式中，发送进程可以将消息发送到缓存中，并立即继续执行，而不必等待接收进程接收到消息。
* 同步通信模式（Synchronous Mode）：在同步通信模式中，发送进程通过MPI\_Ssend函数向接收进程发送消息，并由MPI\_Recv函数接收消息。在同步通信模式中，发送进程会一直阻塞，直到接收进程成功接收到消息并发送一个确认信号。
* 就绪通信模式（Ready Mode）：在就绪通信模式中，发送进程通过MPI\_Rsend函数向接收进程发送消息，并由MPI\_Recv函数接收消息。在就绪通信模式中，发送进程会假设接收进程已经准备好接收消息，因此不会等待接收进程的确认信号。如果接收进程没有准备好接收消息，则会发生错误。



标准通信模式：

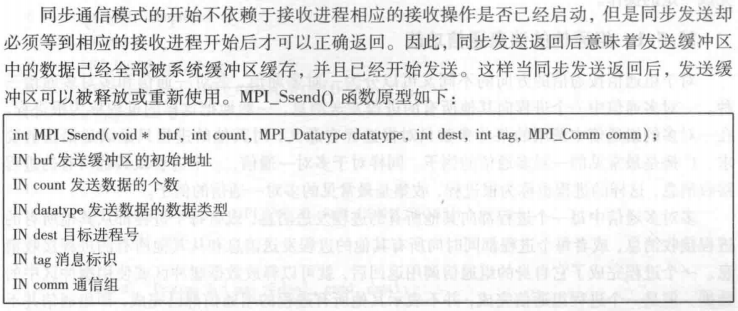


缓存通信模式：

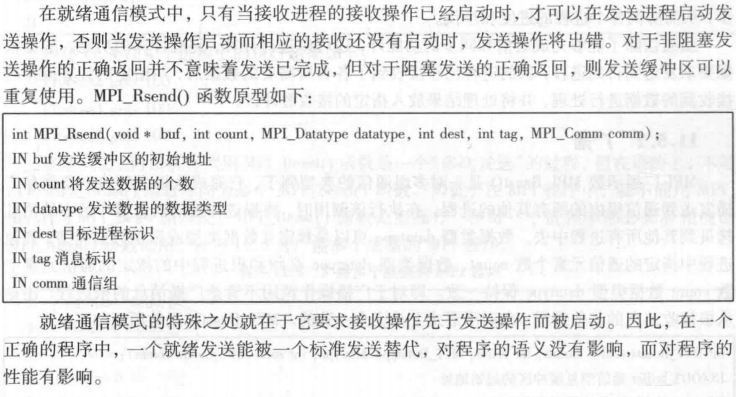




同步通信模式：



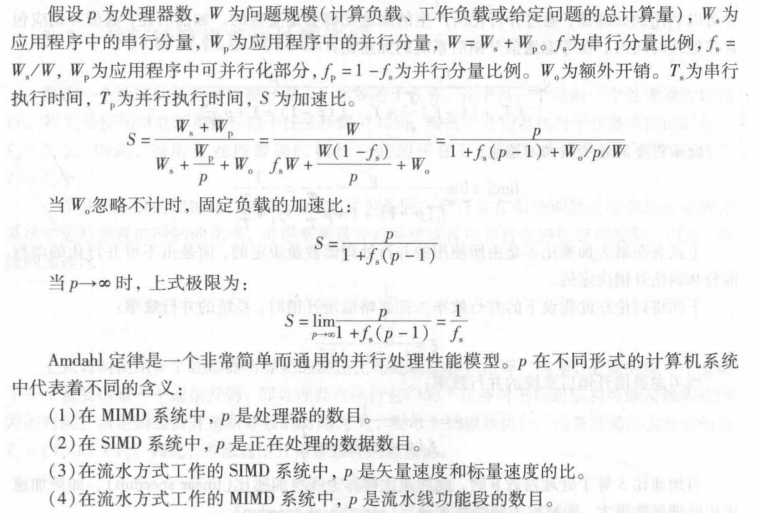
就绪通信模式：



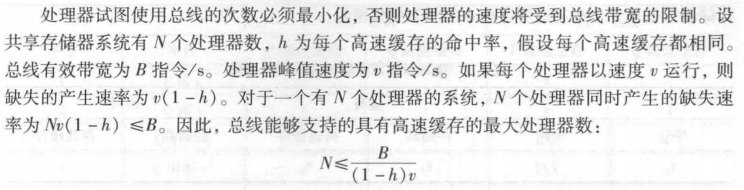
计算题

1. Amdahl定理

程序的结构对加速比的影响是很大的，特别是在并行化中，需要考虑的因素包括数据依赖性、负载平衡、通信开销等。如果程序的可并行代码占比较低，就需要更多的时间在顺序部分上，这将限制加速比的提高。如果程序的并行部分存在数据依赖性，就需要在处理器之间传输数据，这将增加通信开销，降低加速比。此外，如果程序的并行部分存在负载不平衡，即有些处理器的工作负载较重，有些处理器的工作负载较轻，这将降低加速比的提高。因此，在并行化程序时，需要综合考虑以上因素，以最大化加速比的提高。



1. SMP



算法（主要是回答问题，让你讲算法的实现与主要思想）：

数组求和(P138)

奇偶排序(P140)

快速排序(P142)

并行正则排序(P144)

矩阵运算cannon(P146)

矩阵运算fox(P148)

雅可比迭代法(P155)。

数组求和算法：该算法的主要思想是将数组按照固定的大小进行划分，每个线程负责处理一部分数据，然后将各个线程计算出的结果进行合并得到最终结果。具体实现时，可以使用分治思想，将数组划分为多个小数组，然后对这些小数组进行求和，最后将结果合并即可。

**奇偶排序算法：**

1.奇偶排序实际上在多处理器的环境中很有用，处理器可以分别同时处理每一个奇数对，然后又同时处理偶数对。因为奇数对是彼此独立的，每一刻都可以用不同的处理器比较和交换。这样可以非常快速的排序

2.所有的比较和交换是没有数据相关性。因此，每一次比较和交换都可独立执行，也就可以并行化了。比如，第一次奇阶段排序中，100个元素可以分为4个CPU执行，第一个排序1-25，第二个排序26-50，第三个排序51-75，第四个排序76-100。

3. **步骤如下：**

* 选取所有奇数列的元素与其右侧相邻的元素进行比较，将较小的元素排序在前面；
* 选取所有偶数列的元素与其右侧相邻的元素进行比较，将较小的元素排序在前面；
* 重复前面两步，直到所有序列有序为止

**快速排序算法：**

该算法的主要思想是通过选取一个基准值，将待排序的数据分成小于基准值和大于基准值两部分，然后对这两部分分别进行递归排序，最后将排序后的小数组合并起来得到最终的有序序列。在并行实现时，可以直接把左右序列分别交给两个不同的进程递归执行。但是这样会导致进程数量爆炸，所以可以使用任务队列或者进程池来动态管理进程数。

并行正则排序算法：

矩阵运算cannon算法：

矩阵运算fox算法：

雅可比迭代法算法：

MPI基本编程：

1. 打印进程数和进程编号(P194)
2. 简单的发送和接收消息的MPI程序(P196)
3. 广播单个数据的MPI程序(P203)
4. 广播多个数据的MPI程序(P203)
5. 使用归约操作计算Π的MPI程序(P210)
6. 使用蒙特·卡罗方法估计Π的MPI程序(P211)
7. 计算ln2的MPI程序(P212)

OpenMP并行程序设计：

1. 打印总线程数和线程编号OpenMP程序(P278)
2. 打印“Hello，The World！”的OpenMP程序(P283)
3. 使用num\_threads子句指定使用线程数OpenMP程序(P283)
4. 使用矩形法中的中点法计算Π的OpenMP程序(P284)
5. **使用并行域并行化计算Π的OpenMP程序(P284)**
6. 单独使用for指令OpenMP程序(P287)
7. for指令与parallel指令结合起来使用的另一种形式OpenMP程序(P288)
8. 使用for指令并行化计算Π的OpenMP程序(P288)
9. **利用公式计算ln2值的OpenMP程序(P290)**
10. 使用sections结构例子OpenMP程序(P292)
11. 使用sections结构例子的另一种形式OpenMP程序(P293)
12. 使用sections结构向量运算OpenMP程序(P294)
13. 使用private子句的例子OpenMP程序(P296)
14. 使用firstprivate子句的例子OpenMP程序(P297)
15. 使用lastprivate子句的OpenMP程序(P298)
16. 使用threadprivate指令的OpenMP程序(P299)
17. 奇偶排序OpenMP程序(P301)
18. 使用并行归约的并行化计算Π的OpenMP程序(P303)
19. 使用循环制导条件并行归约的并行化计算Π的OpenMP程序(P304)
20. 使用copyin子句的例子OpenMP程序(P304)
21. 使用copyprivate子句的例子OpenMP程序(P305)
22. 使用蒙特·卡罗方法估计Π值的OpenMP程序(P307)
23. 使用private子句和critical指令并行化计算Π的OpenMP程序(P309)
24. 使用atomic指令一个数据进行原子操作保护OpenMP程序(P311)
25. 使用barrier指令进行同步的OpenMP程序(P311)
26. 使用nowait子句除去循环后同步OpenMP程序(P312)
27. 使用master指令的OpenMP程序(P313)
28. 使用ordered指令的OpenMP程序(P315)
29. 使用简单互斥锁计算Π的OpenMP程序(P316)
30. 使用简单互斥锁保护代码区域OpenMP程序(P317)
31. 不使用chunk\_size参数static调度OpenMP程序(P320)
32. 使用chunk\_size参数static调度OpenMP程序(P320)
33. 不使用chunk\_size参数dynamic调度OpenMP程序(P321)
34. 使用chunk\_size参数dynamic调度OpenMP程序(P322)
35. guided调度OpenMP程序(P322)
36. 注意课后P324第三题和第四题，第三题可以利用P290的例子写，第四题可以当作积分的例子，也可以结合P284写积分程序。