[概念 2](#_Toc507661227)

[互连网络 3](#_Toc507661228)

[通信机制 11](#_Toc507661229)

[存储管理 15](#_Toc507661230)

[同步 24](#_Toc507661231)

[多处理机结构 32](#_Toc507661232)

[大规模并行处理机 34](#_Toc507661233)

[机群系统 45](#_Toc507661234)

[多核系统 52](#_Toc507661235)

## 概念

|  |  |
| --- | --- |
| 线路交换 | 源结点和目的结点之间的物理通路在整个数据传送期间一直保持连接 |
| 分组交换 | 把信息分隔成许多组(包)，将它们分别送入互联网路，这些数据包可以通过不同的路径传送，到目的结点后再拼合成原来的数据，结点之间不存在固定连接的物理通路 |
| 网络规模 |  |
| 结点度 |  |
| 结点距离 |  |
| 网络直径 |  |
| 等分带宽 |  |
| 对称网络 |  |
| 静态互联网络 |  |
| 动态互联网络 |  |
| SMP | 集中式共享多处理机(对称式共享多处理机)  由几十个处理器构成，各处理器共享一个集中式的物理存储器，这个存储器相对于各处理器的关系是对称的 |
|  | 分布式共享多处理机 |

## 互连网络

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 互联网络 | 开关元件按照一定的拓扑结构和控制方式构成的网络，实现计算机系统中结点(设备)之间的相互连接。 | | | | | | | | | |
| 要素 | 开关元件 | |  | | | | | | |
| 拓扑结构 | | 互连网络为输入结点到输出结点之间的一组互连或映射 | | | | | | |
| 控制方式 | |  | | | | | | |
|  | 静态互连网络 | | 各结点之间有固定的连接通路，且在运行中不能改变 | | | | | | |
| 动态互联网络 | | 设置有源开关，能根据需要借助控制信号对连接通路加以重新组合，实现所要求的通信模式。 | | | | | | |
| 互连函数 | 表示互联网络中各结点之间连接关系：结点x ↔结点f(x)  反映互联网络输入数组和输出数组之间对应的置换关系或排列关系，因此有时也称为置换函数或排列函数  用二进制表示为[]，n=，N是结点数  双向互连：f(a)=b，f(b)=a  循环函数表示：()()→f()=，f()=，f()=，f()=，f()= | | | | | | | | | |
| 恒等函数 | | 同号结点互联：f()= | | | | | | | |
| 交换函数  (立方体函数) | | 用于：立方体互连网络、超立方体互连网络。 | | | | | | | |
| ：f= | | | | ：f=f | | ：f= | |
|  | |  | | |  | |  | |  |
|  | | 原函数 | | | 第k子函数 | | 第k超函数 | | 逆函数 |
| 均匀洗牌函数 | |  | | |  | |  | |  |
| 蝶式函数 | |  | | |  | |  | |  |
| 反位序函数 | |  | | |  | |  | |  |
| 移数函数 | | (x±k) mod N，k∈[0,N-1] | | | | | | | |
| PM2I函数 | | (x±) mod N，i∈[0,n-1] | | | | | | | |
| 结构  参数 | 网络规模 | | N=互联网络能连接的部件的数量，N越大，网络连接能力(可扩展性)越强 | | | | | | | |
| 结点度 | | d(某结点)=结点所连接的边数，有向网络中，d=入度(进入结点的边数)+出度(从结点出来的边数) | | | | | | | |
| 结点距离 | | 连接两结点的边数最小值 | | | | | | | |
| 网络直径 | | D=最大结点距离，D应尽可能的小 | | | | | | | |
| 等分宽度 | | b=B×w，其中w是通道宽度，单位bit  把由N个结点构成的网络切成结点数相同的两半，在各种切法中，沿切口边数的最小值称为该网络的等分宽度  B反映了网络的最大流量 | | | | | | | |
| 对称性 | | 从任意结点来看，网络的结构都是相同的，实现容易，编程容易 | | | | | | | |
| 性能  指标 | 通信时延 | | 从源结点到目的结点传送一条消息所需的总时间，它由以下4部分构成 | | | | | | | |
| 软件开销 | | 在源结点和目的结点用于收发消息的软件所需的执行时间。  主要取决于两端端结点处理消息的软件内核 | | | | | |
| 通道时延 | | 通过通道传送消息所花的时间。  通路时延 = 消息长度/通道带宽  通常由瓶颈链路的通道带宽决定。 | | | | | |
| 选路时延 | | 消息在传送路径上所需的一系列选路决策所需的时间开销。  与传送路径上的结点数成正比。 | | | | | |
| 竞争时延 | | 多个消息同时在网络中传送时，会发生争用网络资源的冲突。为避免或解决争用冲突所需的时间就是竞争时延。  很难预测，它取决于网络的传输状态 | | | | | |
| 网络时延 | | 通道时延与选路时延的和。  它是由网络硬件特征决定的，与程序行为和网络传输状态无关。 | | | | | | | |
| 端口带宽 | | 对于互连网络中的任意一个端口来说，其端口带宽是指单位时间内从该端口传送到其他端口的最大信息量。  在对称网络中，端口带宽与端口位置无关。网络的端口带宽与各端口的端口带宽相同。  非对称网络的端口带宽则是指所有端口带宽的最小值。 | | | | | | | |
| 聚集带宽 | | 网络从一半结点到另一半结点，单位时间内能够传送的最大信息量。  例如，HPS是一种对称网络  网络规模N的上限：512  端口带宽：40MB/s  HPS的聚集带宽：（40MB/s×512）/2＝10.24GB/s | | | | | | | |
| 等分带宽 | | 与等分宽度对应的切平面中，所有边合起来单位时间所能传送的最大信息量 | | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
| 线性阵列  linear array |  | 总线网络：通过切换来由其所连接的结点分时共享使用  线性阵列：允许多个“源结点-目的结点”对并行使用不同的通道 | d(中间结点)=2  d(端结点)=1  D=N-1  l=N-1  b=1  不对称 |  |
| 环  ring |  | 用一条附加链路将线性阵列的两个端点连接起来而构成。可以单向工作，也可以双向工作 | d=2  D=N/2  l=N  b=2  对称 |  |
| 带弦环  chordal  ring |  | 全连接compeletely connected  d=N-1  D=1  l=N(N-1)/2  b=  Y | 增加的链路愈多，结点度愈高，网络直径就愈小 |  |
| 循环  移树  网络  barrel  shifter |  | 环，每个结点到所有与其距离为2的整数幂的结点连接：  ｜j-i｜=2r(r=0,1,2,…,n-1，n=log2N)，则结点i与结点j连接。 | d=2n－1  N=2n  D=n/2 |  |
| bitree |  | k层完全平衡二叉树 | N=2k-1  d=1~3  D=2(k-1)  b=1  l=N-1  不对称 |  |
| [星形](file:///F:\计算机\Architecture\7-4-13.exe)  star |  | 结点度较高，为N－1。  可靠性差：中心结点故障，则整个系统瘫痪。 | D=2  b=⎣N/2⎦  d=N-1  D=2  l=N-1  不对称 |  |
| fat  tree |  |  |  |  |
| 2D  网格 |  |  | N=n×n  d(内部结点)=4  d(边结点)=3  d(角结点)2  D=2(n-1)  b=n  l=2N-2r  不对称 |  |
| Illiac  网络 |  | 2D网格  每一列两个端结点连接  每一行尾结点与下一行头结点连接  最后一行尾结点与第一行头结点连接 | N=n×n  d=4  D=n-1  b=2n  l=2N  不对称 | PM  PM |
| 2D  [环网](file:///F:\计算机\Architecture\7-4-16.exe) |  | 2D网格  每一行两个端结点连接  每一列两个端结点连接  将环形和网格形组合在一起，并能向高维扩展。 | D=n×n  d=4  D=2×⎣n/2⎦  b=2n |  |
| 超立方体  super cube |  | 二元n-立方体结构  二元n-立方体由N=2n 个结点组成，它们分布在n维上，每维有两个结点。  例 8个结点的3维立方体  4维立方体  为实现一个n-立方体，只要把两个（n－1）立方体中相对应的结点用链路连接起来即可。共需要2n-1条链路。  n-立方体中结点的度都是n，直径也是n，等分宽度为b=N/2 。 | d=n  D=n  l=nN/2  b=N/2  对称 |  |
| 带环  立方体  k-CCC | 1002-2-ok3 | k-立方体中  k个结点构成的环取代每个结点 | N=k×2k  D=2k-1+⎣k/2⎦  l=3N/2  b=N/2k  对称 |  |
| k-n  立方体 | 1001-ok | 环形、网格、环网形、二元n-立方体（超立方体）和Omega网络都是k元n-立方体网络系列的拓扑同构体。  n：是立方体的维数  k：基数，即每一维上的结点数。  k元n-立方体的结点可以用基数为k的n位地址A =a1a2…an来表示。  其中ai表示该结点在第i维上的位置  通常把低维k元n-立方体称为环网，而把高维k元n-立方体称为超立方体。 | N=  d=2n  D=n(k/2)  l=nN  b=2  Y |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 总线  网络 | 构成 | 由一组导线和插座构成 | | |
| 用于 | 实现计算机系统中处理机模块、存储模块和外围设备等之间的互连。 | | |
| 特点 |  | | |
| 工作 |  | | 每一次总线只能用于一个源(主部件)到一个或多个目的(从部件)之间的数据传送。  多个功能模块之间的争用总线或时分总线  总线连接的多处理机系统：  系统总线在处理机、I/O子系统、主存储器以及辅助存储设备（磁盘、磁带机等）之间提供了一条公用通路。  系统总线通常设置在印刷电路板底板上。处理器板、存储器板和设备接口板都通过插座或电缆插入底板 |
| 特点 | 结构简单、成本低、带宽较窄 | | |
| 问题 | 解决总线带宽较窄问题：  1.多总线  (1)为不同的功能设置专门的总线  (2)重复设置相同功能的总线  2.多层次：按层次的架构设置速度不同的总线，使得不同速度的模块有比较适合的总线连接 | | |
| 交叉  开关  网络 |  | |  | |
| 7-12 | | 16×16的交叉开关网络连接把16台处理机与16个存储模块  最多可同时实现16台处理机对16个不同存储模块的并行访问  每个存储模块一次只能满足一台处理机的请求  当多个请求要同时访问同一存储模块时，交叉开关就必须分解所发生的冲突，每一列只能接通一个交叉点开关。  为了支持并行（或交叉）存储器访问，可以在同一行中接通几个交叉点开关。  交叉点开关能在对偶（源、目的）之间形成动态连接，同时实现多个对偶之间的无阻塞连接。  一个n×n的交叉开关网络，可以无阻塞地实现n!种置换。  对一个n×n的交叉开关网络来说，需要n×n套交叉点开关以及大量的连线。 | |
| 特点 | | 带宽和互连特性最好，硬件数量巨大。成本高昂 | |
|  | |  | |

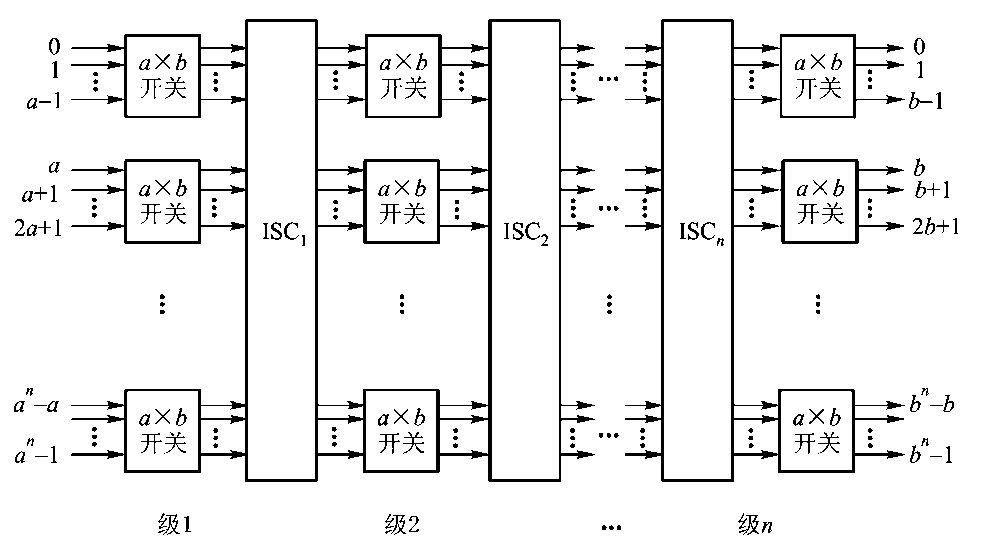
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 多级  互联  网络  MIN | 用于：MIMD、SIMD  各种多级互连网络的区别：开关模块、控制方式、级间互连模式 | | | | | |
| 开关  模块 | [2×2开关的4种连接方式](file:///F:\计算机\Architecture\7-4-22.exe) | | | | |
| 控制  方式 | 对各个开关模块进行控制的方式。  级控制：每一级的所有开关只用一个控制信号控制，只能同时处于同一种状态。  单元控制：每一个开关都有一个独立的控制信号，可各自处于不同的状态。  部分级控制：第i级的所有开关分别用i＋1个信号控制，0≤i≤n－1，n为级数。 | | | | |
| 级间  互连  模式 | 均匀洗牌、蝶式、多路洗牌、纵横交叉、立方体连接等 | | | | |
|  | 开关模块 | 控制方式 | | | 级间互连模式 |
| 多级  立方体 | 2×2开关，二功能(直送和交换)  输入数：N  开关级数：  每级开关模块数：N/2  当第i级(0≤i≤n-1)交换开关处于交换状态时，实现的是Cubei互连函数。 | STARAN网络 | 间接二进制  n方体网络 | |  |
| 级控制  部分级控制 | 单元控制 | |
| Omega | 一个8×8的Omega网络，每级由4个**4功能**的2×2开关构成  级间互连采用均匀洗牌连接方式 | | | k×k开关，N输入，Omega网络  开关级数：  每级开关模块数：N/k  每个开关模块均采用**单元控制方式**。  不同的开关状态组合可实现各种置换、广播或从输入到输出的其他连接。  N=8的多级立方体互连网络的另一种画法  一个k×k开关的合法连接有：  一对一置换连接：一个输入端连接一个输出端  一对多选播连接：一个输入端连接多个输出端  一对多广播连接：一个输入端连接全部输出端  k个输出端被连接到输入端的合法连接数量=  一个k×k开关有k种一对一连接  所欲开关都是一对一连接时才能实现一种一次使用网络的无冲突置换连接  连接函数：  网络可实现的置换连接数即为N个输出端的不同排序的排序数，即N！，所以Omega网络使用一次实现的无冲突置换连接数站可以实现的置换连接数的比例= | |

构成：[a×b](file:///F:\计算机\Architecture\7-4-21.exe)开关模块、级间连接

每一级都用了多个a×b开关

a个输入和b个输出，一般令a＝b＝，k≥1。

相邻各级开关之间都有固定的级间连接



几种常用的开关模块

|  |  |  |
| --- | --- | --- |
| 模块大小 | 合法状态 | 置换连接 |
| 2×2 | 4 | 2 |
| 4×4 | 256 | 24 |
| 8×8 | 16 777 216 | 40 320 |
| n×n | nn | n! |

多级立方体网络

一个8个入端的多级立方体网络:



交换：将有序的一组元素头尾对称地进行交换。

例如：对于由8个元素构成的组，各种基本交换的图形：



STARAN网络：采用级控制(实现交换功能)和部分级控制(实现移数功能)

间接二进制n方体网络：采用单元控制，具有更大的灵活性。

3级STARAN网络在各种级控制信号的情况下所实现的入出端连接以及所实现的交换函数和功能。

控制信号：K2k1k0，ki为第i级的级控制信号。

从表中可以看出下面的4行中每一行所实现的功能可以从级控制信号为其反码的一行中所实现的功能加上1组8元变换来获得。例如：级控制信号为110所实现的功能是其反码001所实现的4组2元交换再加上1组8元交换来获得。

|  |  |  |  |
| --- | --- | --- | --- |
| 级控制信号  k2k1k0 | 连接的输出端号序列  入端号序列：01234567 | 实现的分组交换 | 实现的互连函数 |
| 000 | 0 1 2 3 4 5 6 7 | 恒等 | I |
| 001 | 1 0 3 2 5 4 7 6 | 4组2元交换 | Cube0 |
| 010 | 2 3 0 1 6 7 4 5 | 4组2元交换＋  2组4元交换 | Cube1 |
| 011 | 3 2 1 0 7 6 5 4 | 2组4元交换 | Cube0＋Cube1 |
| 100 | 4 5 6 7 0 1 2 3 | 2组4元交换＋  1组8元交换 | Cube2 |
| 101 | 5 4 7 6 1 0 3 2 | 4组2元交换＋  2组4元交换＋  1组8元交换 | Cube0＋Cube2 |
| 110 | 6 7 4 5 2 3 0 1 | 4组2元交换＋  1组8元交换 | Cube1＋Cube2 |
| 111 | 7 6 5 4 3 2 1 0 | 1组8元交换 | Cube0＋Cube1＋Cube2 |

当STARAN网络用作移数网络时，采用部分级控制，控制信号的分组和控制结果。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 部分级控制信号 | | | | | | 连接的输出端号序列  （入端号序列：01234567） | 所实现的移数功能 |
| 第0级 | 第1级 | | 第2级 | | |
| A  B  C  D | E  G | F  H | I | J | K  L |
| 1  0  0  1  0  1  0 | 1  1  0  1  1  0  0 | 0  1  0  0  1  0  0 | 1  1  1  0  0  0  0 | 0  1  1  0  0  0  0 | 0  0  1  0  0  0  0 | 1 2 3 4 5 6 7 0  2 3 4 5 6 7 0 1  4 5 6 7 0 1 2 3  1 2 3 0 5 6 7 4  2 3 0 1 6 7 4 5  1 0 3 2 5 4 7 6  0 1 2 3 4 5 6 7 | 移1 mod 8  移2 mod 8  移4 mod 8  移1 mod 4  移2 mod 4  移1 mod 2  不移 全等 |

|  |  |  |  |
| --- | --- | --- | --- |
| 网络特性 | 总线系统 | 多级网络 | 交叉开关 |
| 单位数据传送  最小时延 | 恒定 | O(logkn) | 恒定 |
| 每台处理机带宽 | O(w/n)至O(w) | O(w)至O(nw) | O(w)至O(nw) |
| 连线复杂性 | O(w) | O(nwlogkn) | O(n2w) |
| 开关复杂性 | O(n) | O(nlogkn) | O(n2) |
| 连接特性  寻径性能 | 一次只能一对一 | 只要网络不阻塞，  可实现某些置换和广播 | 全置换，  一次一个 |
| 典型计算机 | Symmetry S1，  Encore Multimax | BBNTC-2000  IBM RP3 | Cray Y-MP/816  Fujitsu VPP 500 |
| 说明 | 总线上假定有n台处  理机；总线宽度为w位 | n×n MIN采用k×k  开关，其线宽为w位 | 假定n×n交叉  开关的线宽为w位 |

## 通信机制

|  |  |  |
| --- | --- | --- |
| 共享  存储器  通信 | 组织方案 | 物理上分离的所有存储器，逻辑上统一编址，构成一个逻辑共享存储空间  任一处理器都可以访问该共享空间中的任何一个单元  不同处理器上的同一个物理地址指向的是同一个存储单元 |
| 通信机制 | 处理器之间是通过用load和store指令对相同存储器地址进行读/写操作来实现的。  处理器之间通过load和store指令对相同的存储器地址进行读/写操作  访存时间取决于数据在存储器中的存放位置 |
| 优点 | 与常用的对称式多处理机使用的通信机制兼容。  易于编程，同时在简化编译器设计方面也占有优势。  采用大家所熟悉的共享存储器模型开发应用程序，而把重点放到解决对性能影响较大的数据访问上。  当通信数据量较小时，通信开销较低，带宽利用较好。  可以通过采用Cache技术来减少远程通信的频度，减少了通信延迟以及对共享数据的访问冲突。 |
|  |  |
| 消息  传递 | 组织方案 | 物理上分离的所有存储器，各自有独立的地址空间  任一处理器只能访问自己的本地存储器，不能直接访问其他存储器  不同处理器上的同一个物理地址指向的是不同的存储单元 |
| 通信机制 | 通过处理器间显式地传递消息，这些消息请求进行某些操作或者传送数据。 |
|  | 优点：  1.硬件简单  2.通信显式，因此更容易搞清楚何时发生通信以及通信开销是多少，显式通信可以让编程者重点注意并行计算的主要通信开销，使之有可能开发出结构更好、性能更高的并行程序。  同步很自然地与发送消息相关联，能减少不当的同步带来错误的可能性。 |
|  | 在消息传递的硬件上支持共享存储器就困难得多。所有对共享存储器的访问均要求操作系统提供地址转换和存储保护功能，即将存储器访问转换为消息的发送和接收。  在共享存储器上支持消息传递相对简单  消息传递机制的硬件上实现共享存储通信：将存储器的访问转化为消息的发送和接收 |

消息传递

|  |  |
| --- | --- |
| 格式 | L：包长，单位bit  ：信息包寻径头部的长度 |

|  |  |  |
| --- | --- | --- |
| 通信  模式 | unicast | 一对一 |
| multicast | 一对多 |
| broadcast | 一对全体 |
| conference | 多对多 |

当源结点和目的结点之间没有直接的连接时，消息需要经过中间的结点进行传递。寻径就是用来实现这种传递的通信方法和算法。有的称之为路由

|  |  |
| --- | --- |
| 缓冲区 | 每个结点有各自的缓冲区 |
| 虚拟通道 | 物理通道不变，变换通道两端通信的结点，构成不同的虚拟通道  物理通道由所有虚拟通道分时共享 |
| 死锁 | 缓冲区或通道上的循环等待会引起死锁 |
| 避免死锁 | 增加虚拟同道：增加两条虚拟通道V3和V4  增加虚拟通道可能会使每个请求可用的有效通道带宽降低，为此，当实现数目很大的虚拟通道时需要用高速的多路选择开关 |

|  |  |  |
| --- | --- | --- |
| 寻径  方式 | 线路交换  circuit  switch | 传递消息前，先建立一条从源结点到目的结点的物理通路  传输延时：T=[L+(D+1)]/B  L：包长，单位bit  ：建立路径所需的小包的长度  ：片长  D：中间结点个数  B：带宽  优点：传输带宽大，平均传输时延小，使用的缓冲区小。  缺点：需要频繁地建立源结点到目的结点的物理通路，时间开销大  适用：动态、突发性、大规模、并行处理数据 |
| 存储转发  store  forward | 包是信息基本单位，一次传输一个完整的包  从源结点经过一系列中间结点到达目的结点  每个结点设置包缓冲器，某中间结点收到一个包，保存在包缓冲器，出口链路空闲&下一个结点的包缓冲器空闲→传递给下一个结点  传输延时：T=[L(D+1)]/B  缺点：包缓冲区大，不利于VLSI实现；网络时延大 |
| 虚拟直通  virtual  through | 包是信息基本单位，从源结点经过一系列中间结点到达目的结点  每个结点设置包缓冲器，某中间结点收到一个包，不必等到整个包全部放入缓冲器后再作路由选择，只要接收到用作寻径的包头，就可作出判断，  出口链路空闲→传递给下一个结点  信息包不必存储在该结点的缓冲器中，而是立即传送到下一个结点。  如果整条链路都空闲，包就可以立即直达目的结点。  在输出链路不空闲时，要用缓冲器进行存储。  通信时延：T= [L+(D+1)]/B，一般L>>(D+1)，所以T≈L/B。  当出现寻径阻塞时，虚拟直通方式需要将整个信息包全部存储在寻径结点中，要求每个结点都有足够大的缓冲区。  对存储转发方式的一种改进，减少了网络时延。 |
| wormhole  虫蚀方式 | 一次传输一个完整的包，片是信息基本单位，  包中各片的传送按流水方式进行。  当一个结点把头片送到下一个结点后，那么接下来就可以把后面的各个片也依次送出，一个结点一旦开始传送一个包中的头片后，这个结点就必须等待这个包的所有片都送出去后，才能传送其他包。不同包的片不能混合在一起传送  通信时延：T= D+L/B=(L+D)/B，一般L>>D，所以T≈L/B  出口链路忙时，结点把一个片存储到缓冲器，因此只需小容量的缓冲器  优点：每个结点中缓冲器容量较小，传送延迟小，通道共享性好，利用率高，易于实现选播和广播通信模式。  缺点：当消息的一片被阻塞时，整个消息的所有片都将被阻塞在所在结点，占用了结点资源。 |

|  |  |  |  |
| --- | --- | --- | --- |
| 流控制 | 多个包在某结点为竞争缓冲区或通道资源发生冲突时，使用预先确定好的策略来解决冲突，要使用不会引起用激活死锁的控制网络流量的策略 | | |
|  |  | |
| 包冲突 | 当两个包到达同一个结点时，它们可能都在请求同一个接收缓冲器或者同一个输出通道，这时必须对两个问题进行仲裁。  把通道分配给哪个包？  如何处理被通道拒绝的包？ | |
| 缓冲  暂存 | 把第二个包暂存在缓冲区  优点:不会浪费已经分配了的资源，但它要求结点中有一个足够大的缓冲器来存放整个信息包。 |
| 阻塞  等待 | 阻塞第二个包 |
| 丢弃  重发 | 丢弃第二个包  有可能会造成严重的资源浪费，而且要求重新进行被丢弃包的传输与确认 |
| 阻塞  绕道 | 在包寻径方面提供了更多的灵活性，但为了到达目的结点，可能要花费超过实际需要的通道资源，造成浪费 |
|  |  | |
|  |  | |

流控制策略

两相邻结点之间传送一个片，要同时具备3个条件：

源缓冲区已存有该片&通道已分配好&接收缓冲区准备接收该片。

|  |  |
| --- | --- |
| 确定性寻径 | 预先唯一确定，任意两个结点之间的通信路径  输入源结点与目标结点，就能得到唯一的通信路径，与网络状态无关 |
| 自适应寻径 | 自适应寻径：通信的通路每一次都要根据资源或者网络的情况来选择。  可以避开拥挤的或者有故障的结点，使网络的利用率得到改进。 |

都是建立在维序概念上的

维序寻径：在多维网中，对后继通道的选择是按照各维的顺序来进行的。

二维网格网络：X-Y寻径：先沿X维方向寻径，再沿Y维方向寻径

超立方体：E-cube寻径：

二维网格网络的X-Y寻径

源节s=(x1，y1)，目的结点d(x2，y2)，从s出发，先沿X轴方向前进，直到找到d 所在的列x2；然后再沿Y轴方向前进，直到找到目标结点（x2，y2）。

选播和广播寻径算法

多计算机网络中会出现以下4种：

|  |  |  |
| --- | --- | --- |
| 通信  模式 | 单播 | 对应于一对一的通信情况，即一个源结点发送消息到一个目的结点 |
| 选播 | 对应于一到多的通信情况，即一个源结点发送同一消息到多个目的结点 |
| 广播 | 对应于一到全体的通信情况，即一个源结点发送同一消息到全部结点 |
| 会议 | 对应于多到多的通信情况 |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

通道流量和通信时延是常用的两个参数。

通道流量可用传输有关消息所使用的通道数来表示。

通信时延则用包的最大传输时间来表示。

优化的寻径网络应能以最小流量和最小时延实现相关的通信模式。

以网格网络为例，讨论选播和广播。

在3×4网格上实现的选播寻径

源结点：S

传送一个包到标号为Di 的5个目的结点(i＝1，2，…，5)

图（a）：目的结点为5个的选播可以用5次单播来实现

X-Y寻径的流量需要用1＋3＋4＋3＋2＝13条通道。

到D3的路径最长，所以时延是4。

图（b）和（c）给出了两种选播寻径模式，流量分别为7和6。

在虫蚀网络中，用图（c）的选播寻径模式比较好。

在存储转发网络中，则用图（b）的寻径模式比较好，时延较短。

图（d ）：使用一棵4层的生成树可以把一个包从结点S广播到所有的网络结点

产生的时延和流量最小



## 存储管理

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 存储系统 | SMP/UMA  集中式  (对称式)  共享  多处理机 | | 所有处理器共享一个集中式物理存储器，即单一主存，这个主存对于各处理器的关系是对称的  处理器个数较少，，  支持对共享数据和私有数据的Cache缓存。  私有数据：只供一个处理器使用的数据  共享数据：供多个处理器使用  多级Cache可以降低处理器对存储器带宽的要求，如果每个处理器对存储器带宽的要求都降低了，那么多个处理器就可以共享一个存储器。  适用：网络规模较小(几十个处理器)  优点：经济  缺点：可扩放性很差，每当Cache不命中，就要与所有其他的Cache进行通信，这时通过总线的广播以及其他Cache的监听来实现的  使用广播和监听，是因为系统中没有一个集中的数据结构来记录Cache的状态，数据块的状态保存在各自的Cache中，  所以当Cache不命中，它不知道其他处理器的的Cache中哪些拥有相应的副本，就只好在总线上广播，让所有其他Cache都来协助完成相关操作  对于大规模系统，总线广播很多，总线成为系统的瓶颈  目录协议的实现开销比监听式协议的稍微大一些，但可以用于实现更大规模的处理机 | | |
| DSM/NUMA  分布式  共享  多处理机 | | 每个处理机有属于自己的本地存储器，存储器在物理上是分布的，这些分布在各个处理机的存储器统一编址，逻辑上组成一个共享存储器，这些存储器通过互联网络连接，  每个处理机除了可以访问自己的存储器外，还能访问其它处理机的存储器  每个结点包含：处理器、存储器、I／O、互连网络接口  在许多情况下，分布式存储器结构优于集中式共享存储器结构。  簇：超级结点  每个结点内包含个数较少（例如2～8）的处理器；  处理器之间可采用另一种互连技术（例如总线）相互连接形成簇。  优点：如果大多数的访问是针对本结点的局部存储器，则可降低对存储器和互连网络的带宽要求；对本地存储器的访问延迟时间小。  缺点：处理器之间的通信较为复杂，且各处理器之间访问延迟较大。 | | |
|  | |  | | |
| Cache  一致性 | 问题 | 每一个处理器各自拥有一个Cache，共享数据进入Cache，多个处理器的Cache拥有同一存储块的副本  某处理器修改其Cache数据后，和其他处理器的Cache内容不一致 | | | |
| 条件 | 存储系统一致：对某个数据项的任意读操作均可得到其最新写入值：  1.处理器对某存储单元进行一次写后读，写和读之间没有其他处理器对该单元进行写  2.先处理器P写存储单元X，再处理器Q对单元X进行读，写和读之间无其他写  3.写串行化：任意两个处理器对同一单元的两次写，从各个处理器的角度看来顺序都是相同的。 | | | |
| 措施 | 1.共享数据的迁移：减少了对远程共享数据的访问延迟，也减少了对共享存储器带宽的要求。  2.共享数据的复制：不仅减少了访问共享数据的延迟，也减少了访问共享数据所产生的冲突。  一般小规模多处理机采用硬件方法实现Cache一致性。  Cache一致性协议：在多个处理器中用来维护一致性的协议。  **关键：跟踪记录共享数据块的状态** | | | |
| 目录式协议  directory | | | 物理存储器中数据块的共享状态被保存在一个称为目录的地方  用于DSM存储系统 |
| 监听式协议  snooping | | | 每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线（它们一直在监听）来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作  用于SMP存储系统 |
|  |  | |  | |
|  | |  | |
|  | |  | |

监听协议

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 关键 | | 1.使用具有广播功能的互联网络，如总线网络  2.CPU访问共享存储器或需要其他处理器的Cache做相应操作如作废等，时，向总线发送信息  3.所有处理器持续监听总线，当检测到总线上的地址在它们的Cache中有副本，则响应该消息，并进行相应的操作 | | | | |
| 写Cache | | 多个处理器同时写入，竞争成功者获得访问权之后，其他处理器要访问该数据，发生CacheMiss， | | | | |
| 写作废 | 写入Cache时，查看其他处理器的Cache，如果有该副本则删除，本处理器Cache的该副本最新并且唯一  对同一个Cache块进行多次写操作且中间无读操作，仅在第一次写时进行一次作废操作即可  写作废针对Cache块进行操作， | | | |
| 写更新 | 写入Cache时，查看其他处理器的Cache，如果有该副本则更新，本处理器Cache的该副本时最新但不唯一  对同一个Cache块进行多次写操作且中间无读操作，每一次写操作都要进行一次广播  写更新针对字（或字节）进行  延迟时间较小 | | | |
| 写直达  write  through | 写入Cache时写回主存， | | | |
| 写回  write  back | 写入Cache时不写回主存，修改Cache状态→M  写回法得到数据的最新值会困难一些，因为最新值可能在某个Cache中，也可能在主存中  每个处理器的Cache控制器都监听其他处理器放到总线上的地址，如果某个处理器发现它拥有被请求数据块的一个最新副本，他就把这个数据块送给发出请求的处理器  与write through相比，尽管写回法  缺点：复杂度较高； 优点：所需存储带宽低 | | | |
| 处理器  ↓  总线 | | Cache块  地址 |  | | | |
| Cache操作 | RdMiss |  | | |
| WtMiss |  | | |
| Invalidate | 通知其他各处理器作废其Cache中相应的副本  写命中时，先发送Invalidate，再写入 | | |
| Cache块  状态 | 无效I | 本Cache块空闲，  访问时，必然不命中，执行调入  监听时，即使Cache地址匹配，也不执行任何操作 | | |
| 共享S | 本Cache块的内容有效，最新但不唯一(共享)，对应内存块内容相同(Cache一致)  访问时，如果不命中则执行替换  监听时，如果Cache地址匹配， | | |
| 已修改M | 本Cache块内容有效，最新并且唯一；对应内存块内容是过时的(Cache不一致)  访问时，如果不命中则执行替换  监听时，如果Cache地址匹配， | | |
| 每个结点内嵌一个有限状态控制器，根据来自处理器或总线的请求以及Cache块的状态，做出相应的响应 | | | |
| 访问  Cache  **广播** |  | 本处理器目标Cache块状态=I | | | 本处理器目标Cache块状态=S | 本处理器目标Cache块状态=M |
| 读命中 | / | | | 状态S→S | 状态M→M |
| 写命中 | / | | | 发送Invalidate；状态S→M | 状态M→M |
| 读不命中 | 发送RdMiss；调入；状态I→S | | | 发送RdMiss；替换；状态S→S | 发送RdMiss；写回；替换；状态M→S |
| 写不命中 | 发送WtMiss；调入；状态I→M | | | 发送WtMiss；替换；状态S→M | 发送WtMiss；写回；替换；状态M→M |
| **监听**总线 | 总线消息 |  | | |  |  |
| RtMiss | / | | | 状态S→S | 状态M→S  写回，终止消息发送方处理器访问内存  向请求方处理器提供该块，  本处理器再执行写回 |
| WtMiss | / | | | 执行作废：状态S→I | 执行作废：状态M→I  写回，终止消息发送方处理器访问内存 |
| Invalidate | / | | | 执行作废：状态S→I | 执行作废：状态M→I |

读/写不命中时，通过总线找到相应数据块的最新副本，一般是马上启动对存储器相关块的访问，以尽快获得这个副本

所有处理器持续监听总线，当检测到总线消息中的Cache地址在自己的Cache中有匹配的副本，则对该Cache块执行以下操作，目标Cache块状态S或M

否则，不做任何操作

目录协议

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 目录协议 | 完全由硬件实现，或软硬件结合  对于任何一个数据块，都可以快速地在唯一的一个位置中找到相关的信息。这使一致性协议避免了广播操作。  目录协议的实现开销比监听式协议的稍微大一些，但可以用于实现更大规模的多处理器系统  在监听协议中，相关的消息要放到总线上进行广播，目录协议则是由点到点的通信来完成  本地结点把请求发给宿主结点中的目录，再由目录控制器有选择的向远程结点发出相应的消息，使远程结点进行相应的操作，并进行目录中状态等信息的更新。  不同目录协议的区别：1.设置的内存状态及个数 2.共享集结构(目录结构) | | | | | | |
| 目录表 | 每一个处理器对应配置一个目录表，每一个内存块对应一个目录表的表项 | | | | | | |
| 内存块  状态 | 未缓冲U | | | 所有处理器的Cache中都没有这个块的副本 | | |
| 共享S | | | 存在处理机的Cache中都有这个块的副本，且Cache一致 | | |
| 独占E | | | 仅有一个处理机有这个块的副本，且Cache不一致 | | |
| 共享集 | 每一个处理器对应配置一个共享集，共享集用位向量表示：一个二进制数，位数等于系统中处理器个数，每一位对应一个处理器，1→该处理器的Cache中有该内存块副本(不管是否Cache一致)，0→没有 | | | | | |
| 处理机 | 本地结点 | | | 发出访问请求的结点 | | | |
| 宿主结点1 | | | 本地结点访问某内存块，该内存块所属的结点 | | | |
| 宿主结点2 | | | 本地结点替换某Cache块，原Cache块对应的内存块所属的结点 | | | |
| 远程结点 | | | Cache包含请求数据  可以和宿主结点是同一个结点，也可以不是同一个结点 | | | |
| 消息  (点对点) | 消息 | | 方向 | | | 触发 | 执行 |
| RdMiss | | 本地结点→宿主结点 | | | 读不命中 |  |
| WtMiss | | 本地结点→宿主结点 | | | 写不命中 |  |
| Invalidate | | 本地结点→宿主结点 | | | 写命中 | **作废**所有远程结点中的该Cache块(如果有) |
| MdSharer | | 本地结点→旧宿主结点 | | | 替换  某Cache块(S) | 旧宿主结点共享集，删除本地结点，若变为空，则该Cache块对应的内存块状态→U |
| WtBack2 | | 本地结点→旧宿主结点 | | | 替换  某Cache块(M) | **写回**原Cache块  旧宿主结点共享集，删除本地结点，若变为空，则该Cache块对应的内存块状态→U |
| Invalidate2 | | 宿主结点→远程结点 | | |  | 要求远程结点作废，地址对应的Cache块 |
| Fetch | | 宿主结点→远程结点 | | | RdMiss&E | 远程结点收到后返回目标Cache块内容 |
| Fetch&Inv | | 宿主结点→远程结点 | | | WtMiss&E | 远程结点收到后返回目标Cache块内容  远程结点作废该Cache块 |
| DReplay | | 宿主结点→本地结点 | | | RdMiss||WtMiss | 发送目标Cache块，本地结点收到后执行调入或替换 |
| WtBack | | 远程结点→宿主结点 | | | Fetch||Fetch&Inv | **写回**目标Cache块 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 处理器 | 操作 | 目标Cache块原状态 | | | | |
| I | S | | M | |
| 本地结点 | 读命中 | / | S→S | | M→M | |
| 写命中 | / | Invalidate；状态S→M | | M→M | |
| 读不命中 | RdMiss；I→S； | RdMiss；MdSharer；替换S→S | | RdMiss；WtBack2；M→S | |
| 写不命中 | WtMiss；I→M； | WtMiss；MdSharer；替换S→M | | WtMiss；WtBack2；M→M | |
| DReply |  |  | |  | |
| 远程结点 | Fetch | / | / | | WtBack； | |
| Fetch&Inv | / | / | | WtBack；M→I | |
| Invalidate | / | S→I | | / | |
| 宿主结点 |  | 目标Cache块对应的内存块原状态 | | | | |
|  | U | | S | | E |
| RdMiss | DReply；U→S；  共享集={本地结点} | | DReply；S→S  共享集添加本地结点 | | Fetch；DReply；E→S  共享集添加本地结点 |
| WtMiss | DReply；U→E  共享集={本地结点} | | Invalidate；DReply；S→E  共享集={本地结点} | | Fetch&Inv；DReply；E→E  共享集={本地结点} |
| WtBack |  | |  | | 共享集={} |
| Invalidate |  | |  | | 发送WtBack |
| MdSharer | / | | 共享集删除本地结点  S→U(共享集空) 或S→S(非空) | | / |
| WtBack2 | / | |  | | 写回对应内存块  共享集删除本地结点  S→U(共享集空) 或S→S(非空) |

优化：对某个独占快发出读或写不命中时，该块将先被送往宿主结点存入存储器，然后再被送往请求结点，而实际中的计算机很多都是讲数据从拥有者结点直接送给请求结点，同时写回宿主结点中的存储器

|  |  |  |
| --- | --- | --- |
| 目录结构 | 全映像 | 每一个处理器对应配置一个共享集，共享集用位向量表示：一个二进制数，位数等于系统中处理器个数，每一位对应一个处理器，1→该处理器的Cache中有该内存块副本(不管是否Cache一致)，0→没有  优点：处理简单，速度快。  缺点：存储空间开销大，目录项的数目与处理机数成正比，而目录项的大小（位数）也与N成正比，因此目录所占用的空间与N2成正比。  可扩放性很差。当位向量中的值为“1”时，就表示它所对应的处理机有该数据块的副本；否则就表示没有。  在这种情况下，共享集合由位向量中值为“1”的位所对应的处理机构成。 |
| 有限  映像 | 核心思想：采用位数固定的目录项目  限制同一数据块在所有Cache中的副本总数。  例如，限定为常数m。则目录项中用于表示共享集合所需的二进制位数为：m×log2N。  目录所占用的空间与N×成正比。  缺点：当同一数据的副本个数大于m时，必须做特殊处理。当目录项中的m个指针都已经全被占满，而某处理机又需要新调入该块时，就需要在其m个指针中选择一个，将之驱逐，以便腾出位置，存放指向新调入块的处理机的指针 。 |
| 链式 | 用一个目录指针链表来表示共享集  当一个数据块的副本数增加（或减少）时，其指针链表就跟着变长（或变短）。  优点：链表长度不受限制，既不限制副本的个数，又保持了可扩展性。  替换某Cache块时，删除对应的链表元素：  方法1：沿链表往下寻找第i个元素，修改其前后的链接指针，跳过该元素。  方法2：沿链表往下寻找第i个元素，作废它及其后的所有元素所对应的Cache副本。  根据所使用的链表可分为单链式和双链式 |

## 同步

同步机制：硬件提供的同步指令的基础上，通过用户级软件历程来建立

硬件原语：读/写存储单元的原子操作

是同步机制的关键，作为基本构建，构造各种各样的用户级同步操作，这些原语主要是供系统程序员来编制同步库函数

要求一条指令不可中断，该指令完成一次存储器读和存储器写，而且要避免死锁

这很困难

一些计算机采用一对指令而不是一条指令来实现上述同步原语，这种方法在第二条指令返回一个值，通过该值可以判断该指令对的执行结果是否相当于一个原子操作，即所有其他处理器进行的操作或者是该指令对之前、或者是该指令对之后进行，不存在这两条指令之间进行的操作

该指令有两条特殊的指令构成：

load：LL(Load Linked)

store：SC(Store Conditional)

指令顺序执行：如果由LL指明的存储单元的内容在SC对其进行写之前已被其他指令改写过，则第二条指令SC执行失败；

如果在两条指令间进行切换也会导致SC执行失败。

如果执行成功则SC返回1，否则返回0.

LL则返回该存储单元初始值

基本硬件原语(原子操作)：

|  |  |
| --- | --- |
| atomic exchange  原子交换 | 将一个存储单元的值和一个寄存器的值进行交换  建立一个锁，锁值：0可用；1上锁不可用  处理器上锁时，将对应于该锁的存储单元的值与存放在某个寄存器中的1进行交换。如果返回值为0，存储单元的值此时已置换为1，防止了别的进程竞争该锁。  实现同步的关键：操作的原子性 |
| test\_and\_set  测试并置定 | 先测试一个存储单元的值，如果符合条件则修改其值 |
| fetch\_and\_increment  读取并加1 | 返回存储单元的值并自动增加该值。 |
| 使用指令对 | LL(load linked或load locked)的取指令  SC(store conditional)的特殊存指令  指令顺序执行  1.如果由LL指明的存储单元的内容在SC对其进行写之前已被其他指令改写过，则第二条指令SC执行失败；  2.如果在两条指令间进行切换也会导致SC执行失败。  3.SC将返回一个值来指出该指令操作是否成功：1：成功，0：不成功  4.LL则返回该存储单元初始值。  LL/SC机制的一个优点：用来构造别的同步原语  5.指令对的实现必须跟踪地址：由LL指令指定一个寄存器，该寄存器存放着一个单元地址，这个寄存器常称为连接寄存器。  6.LL／SC原语的另一个优点：读写操作明显分开  LL不产生总线数据传送，这使下面代码与使用经过优化交换的代码具有相同的特点：  lockit：LL R2, 0（R1）  BNEZ R2, lockit  DADDIU R2, R0, #1  SC R2, 0（R1）  BEQZ R2, lockit  第一个分支形成环绕的循环体，第二个分支解决了两个处理器同时看到锁可用的情况下的争用问题。尽管旋转锁机制简单并且具有吸引力，但难以将它应用于处理器数量很多的情况。 |
|  |  |

用一致性实现锁

采用多处理机的一致性机制来实现旋转锁。

旋转锁：处理器环绕一个锁不停地旋转而请求获得该锁。适合于这样的场合：锁被占用的时间很少，在获得锁后加锁过程延迟很小。

1.无Cache一致性机制：在存储器中保存锁变量，处理器可以不断地通过一个原子操作请求使用权。

比如：利用原子交换操作，并通过测试返回值而知道锁的使用情况。释放锁的时候，处理器只需简单地将锁置为0。

例：用原子交换操作对旋转锁进行加锁，R1中存放的是该旋转锁的地址。

2.机器支持Cache一致性

将锁调入Cache，并通过一致性机制使锁值保持一致。

优点：可使“环绕”的进程只对本地Cache中的锁（副本）进行操作，而不用在每次请求占用锁时都进行一次全局的存储器访问；

可利用访问锁时所具有的局部性，即处理器最近使用过的锁不久又会使用，减少为获得锁而花费的时间

3.改进旋转锁（获得第一条好处）

只对本地Cache中锁的副本进行读取和检测，直到发现该锁已经被释放。然后，该程序立即进行交换操作，去跟在其他处理器上的进程争用该锁变量。

修改后的旋转锁程序：

lockit： LD R2, 0（R1）

BNEZ R2, lockit

DADDIU R2, R0, #1

EXCH R2, 0（R1）

BNEZ R2, lockit

3个处理器利用原子交换争用旋转锁所进行的操作

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 步骤 | 处理器P0 | 处理器P1 | 处理器P2 | 锁状态 | 总线/目录操作 |
| 1 | 占有锁 | 环绕测试  是否lock=0 | 环绕测试  是否lock=0 | 共享 | 无 |
| 2 | 将锁  置为0 | 收到作废命令 | 收到作废命令 | 专有（P0） | P0发出对  锁变量的作废消息 |
| 3 |  | Cache不命中 | Cache不命中 | 共享 | 总线/目录收到P2  Cache不命中；  锁从P0写回 |
| 4 |  | 因总线/目录  忙而等待 | lock=0 | 共享 | P2 Cache不命中  被处理 |
| 5 |  | Lock=0 | 执行交换，  导致Cache  不命中 | 共享 | P1 Cache不命中  被处理 |
| 6 |  | 执行交换，  导致Cache  不命中 | 交换完毕：返回0  并置lock=1 | 专有（P2） | 总线/目录收到P2  Cache不命中；  发作废消息 |
| 7 |  | 交换完毕：  返回1 | 进入关键程序段 | 专有（P1） | 总线/目录处理P1  Cache不命中；  写回 |
| 8 |  | 环绕测试  是否lock=0 |  |  | 无 |

同步性能问题

简单旋转锁不能很好地适应可缩扩性。大规模多处理机中，若所有的处理器都同时争用同一个锁，则会导致大量的争用和通信开销

本例中问题的根源：锁的争用、对锁进行访问的串行性以及总线访问的延迟。

旋转锁的主要优点：总线开销或网络开销比较低，而且当一个锁被同一个处理器重用时具有很好的性能。

如何用旋转锁来实现一个常用的高级同步原语：栅栏

栅栏强制所有到达该栅栏的进程进行等待，直到全部的进程到达栅栏，然后释放全部的进程，从而形成同步。

栅栏的典型实现

用两个旋转锁：

用来保护一个计数器，它记录已到达该栅栏的进程数；

用来封锁进程直至最后一个进程到达该栅栏。

一种典型的实现

其中：

lock和unlock提供基本的旋转锁

变量count记录已到达栅栏的进程数

total规定了要到达栅栏的进程总数

对counterlock加锁保证增量操作的原子性。

release用来封锁进程直到最后一个进程到达栅栏。

spin（release=1）使进程等待直到全部的进程到达栅栏。

实际情况中会出现的问题

栅栏通常是在循环中使用，从栅栏释放的进程运行一段后又会再次返回栅栏，这样有可能出现某个进程永远离不开栅栏的状况(它停在旋转操作上)。

一种解决方法

当进程离开栅栏时进行计数（和到达时一样），在上次栅栏使用中的所有进程离开之前，不允许任何进程重用并初始化本栅栏。但这会明显增加栅栏的延迟和竞争。

另一种解决办法

采用sense\_reversing栅栏，每个进程均使用一个私有变量local\_sense，该变量初始化为1。

sense\_reversing栅栏的代码

优缺点：使用安全，但性能比较差。

对于10个处理器来说，当同时进行栅栏操作时，如果忽略对Cache的访问时间以及其他非同步操作所需的时间，则其总线事务数为204个，如果每个总线事物需要100个时钟周期，则总共需要20400个时钟周期。

当竞争不激烈且同步操作较少时，我们主要关心的是一个同步原语操作的延迟。

即单个进程要花多长时间才完成一个同步操作。

基本的旋转锁操作可在两个总线周期内完成：

一个读锁

一个写锁

我们可用多种方法改进，使它在单个周期内完成操作。

同步操作最严重的问题：进程进行同步操作的串行化。它大幅度地增加了完成同步操作所需要的时间

同时多线程

线程级并行性

（Thread Level Parallelism，简称TLP）

线程是进程内的一个相对独立且可独立调度和指派的执行单元，它比进程要“轻巧”得多。

只拥有在运行过程中必不可少的一点资源，如：程序计数器、一组寄存器、堆栈等。

线程切换时，只需保存和设置少量寄存器的内容，开销很小。

线程切换只需要几个时钟周期。

进程的切换一般需要成百上千个处理器时钟周期。

实现多线程有两种主要的方法：

细粒度（fine-grained）多线程

在每条指令之间都能进行线程的切换，从而使得多个线程可以交替执行。

通常以时间片轮转的方法实现这样的交替执行，在轮转的过程中跳过当时处于停顿的线程。

CPU必须在每个时钟周期都能进行线程的切换。

主要优点：既能够隐藏由长时间停顿引起的吞吐率的损失，又能够隐藏由短时间停顿带来的损失。

主要缺点：减慢了单个线程的执行

粗粒度（coarse-grained）多线程

线程之间的切换只发生在时间较长的停顿出现时。

例如：第二级Cache不命中。

减少了切换次数，也不太会降低单个线程的执行速度 。

缺点：减少吞吐率损失的能力有限，特别是对于较短的停顿来说更是如此。

原因：由粗粒度多线程的流水线建立时间的开销造成的。由于实现粗粒度多线程的CPU只执行单个线程的指令，因此当发生停顿时，流水线必须排空或暂停。停顿后切换的新线程也有个填满流水线的过程，填满后才能不断地流出指令执行结果。

将线程级并行转换为指令级并行

同时多线程技术

Simultaneous MultiThreading，简称SMT

一种在多流出、动态调度的处理器上同时开发线程级并行和指令级并行的技术。

提出SMT的主要原因

现代多流出处理器通常含有多个并行的功能单元，而单个线程不能有效地利用这些功能单元。

通过寄存器重命名和动态调度机制，来自各个独立线程的多条指令可以同时流出，而不用考虑它们之间的相互依赖关系，其相互依赖关系将通过动态调度机制得以解决。

一个超标量处理器在4种情况下的资源使用情况：

不支持多线程技术的超标量处理器

由于缺乏足够的指令级并行而限制了流出槽的利用率。

支持粗粒度多线程的超标量处理器

通过线程的切换部分隐藏了长时间停顿带来的开销，提高了硬件资源的利用率。

只有发生停顿时才进行线程切换，而且新线程还有个启动期，所以仍然可能有一些完全空闲的时钟周期。

支持细粒度多线程的超标量处理器

线程的交替执行消除了完全空闲的时钟周期。

由于在每个时钟周期内只能流出一个线程的指令，ILP的限制导致了一些时钟周期中依然存在不少空闲流出槽。

支持同时多线程的超标量处理器

在同一个时钟周期中可以让多个线程使用流出槽。

理想情况下，流出槽的利用率只受限于多个线程对资源的需求和可用资源间的不平衡 。

不支持多线程的情况



多线程的3种情况



开发的基础: 动态调度的处理器已经具备了开发线程级并行所需的许多硬件设置。

动态调度超标量处理器有一组很多的虚拟寄存器，可以用作各独立线程的寄存器组。

由于寄存器重命名机制给各寄存器提供了唯一的标识，多个线程的指令可以在数据路径上混合执行，而不会导致各线程之间源操作数和目的操作数的混乱。

多线程可以在一个乱序执行的处理器的基础上实现，只要为每个线程设置重命名表、分别设置各自的程序计数器、并为多个线程提供指令确认的能力。

同时多线程处理器的设计

同时多线程只有在细粒度的实现方式下才有意义

细粒度调度方式会对单个线程的性能产生不利的影响

可以通过采用优先线程的方法来尽可能地减少。这种方法既能保持多线程在性能上的优势，又对单个线程的性能影响比较少。

多个线程的混合执行不可避免地会影响单个线程的执行速度

为提高单个线程的性能，应该为指定的优先线程尽可能多地向前取指（或许在分支指令的两条路径上都要向前取指）；

在分支预测失败和预取缓冲器不命中的情况下清空取指单元。但是这样限制了其他线程可用来调度的指令条数，从而降低了吞吐率。

所有的多线程处理器都必须在这里寻求一种折中方案。

只要一有可能，处理器就运行指定的优先线程。

从取指阶段开始就优先处理优先线程：只要优先线程的指令预取缓冲区未满，就为它们优先取指。

只有当优先线程的缓冲区填满以后才为其他线程预取指令。

当有两个优先线程时，意味着需要并发预取两条指令流，这给取指部件和指令Cache都增添了复杂度。

指令流出单元也要优先考虑指定的优先线程，只有当优先线程停顿不能流出的时候才考虑其他线程。

设计同时多线程处理器时面临的其他主要问题：

需要设置更大的寄存器组，用来保存多个线程的现场。

不能影响时钟周期，特别是在关键路径上

如指令流出和指令完成：

指令流出时，有更多的候选指令需要考虑；

指令完成时，选择提交哪些指令可能会比较困难。

需要保证由于并发执行多个线程带来的Cache冲突和TLB冲突不会导致明显的性能下降。

需要重视的两个实际情况：

在许多情况下，多线程所导致的潜在额外性能开销是很小的，简单的线程切换选择算法就足够好了；

目前的超标量处理器的效率是比较低的，还有很大的改进余地，即使增加一些开销也是值得的

同时多线程处理器的性能

在超标量处理器上增添8个线程的同时多线程能力时获得的性能提高（单位：指令数/每拍）

两个特点

超标量处理器本身功能十分强大，它具有很大的一级Cache、二级Cache以及大量的功能单元。仅仅采用指令级并行，不可能利用全部的硬件性能，因此超标量处理器的设计者不可能不考虑使用诸如同时多线程这样的技术来开发线程级并行。

同时多线程的能力也很强大，可以支持8个线程，并为两个线程同步取指。

将超标量和同时多线程结合起来，在指令级并行基础上进一步开发线程级并行，可以获得显著的性能提高。

## 多处理机结构

|  |  |  |
| --- | --- | --- |
| 多处  理机  结构 | SISD |  |
| SIMD |  |
| MISD |  |
| MIMD | 灵活性好，可以充分利用商品化微处理器在性能价格比方面的优势，已成为通用多处理机系统结构的选择  包括：Cluster(计算机机群系统)、PVP、SMP、MPP、DSM |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | PVP | SMP | MPP | DSM | Cluster |
| 处理器  类型 | 专用定制 | 商用 | 商用 | 商用 | 商用 |
| 互连网络 | 定制  交叉开关 | 总线  交叉开关 | 定制网络 | 定制网络 | 商用网络  以太网、  ATM |
| 通信机制 | 共享变量 | 共享变量 | 消息传递 | 共享变量 | 消息传递 |
| 地址空间 | 单地址  空间 | 单地址  空间 | 多地址  空间 | 单地址  空间 | 多地址  空间 |
| 系统  存储器 | 集中共享 | 集中共享 | 分布非共享 | 分布共享 | 分布非共享 |
| 访存模型 | UMA | UMA | NORMA | NUMA | NORMA |
| 代表机器 | Cray C-90  Cray T-90  NEC SX4  银河1号 | IBM R50，  SGI Power  Challenge，  DEC Alpha  服务器8400  曙光1号 | Intel Paragon，  IBM SP2，  Intel TFLOPS，  曙光1000/2000 | Stanford DASH，  Cray T 3D，  SGI/Cray  Origin 2000 | Berkeley NOW，  Alpha Farm，  Digital Trucluster |

|  |  |
| --- | --- |
| 计算/通信比 | 在并行计算中，每次数据通信要进行计算域通信开销的比值 |
| 并行处理面临的挑战 | 程序中的并行性有限  相对较大的通信开销  第一个挑战  有限的并行性使计算机要达到很高的加速比十分困难  第二个挑战：多处理机中远程访问的延迟较大  在现有的机器中，处理器之间的数据通信大约需要50～1000个时钟周期。  主要取决于：  通信机制、互连网络的种类和机器的规模  问题的解决：  并行性不足： 采用并行性更好的算法  远程访问延迟的降低：靠系统结构支持和编程技术  在并行处理中，影响性能（负载平衡、同步和存储器访问延迟等）的关键因素常依赖于：  应用程序的高层特性  如数据的分配，并行算法的结构以及在空间和时间上对数据的访问模式等。  依据应用特点可把多机工作负载大致分成两类：  单个程序在多处理机上的并行工作负载  多个程序在多处理机上的并行工作负载  并行程序的计算／通信比率  反映并行程序性能的一个重要的度量：  计算与通信的比率  计算／通信比率随着处理数据规模的增大而增加；随着处理器数目的增加而减少 |

在几种不同的共享存储器并行计算机中远程访问一个字的典型延迟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 机器 | 通信机制 | 互连网络 | 处理机  最大数量 | 典型远程存储器  访问时间（ns） |
| Sun Starfire servers | SMP | 多总线 | 64 | 500 |
| SGI Origin 3000 | NUMA | 胖超立方体 | 512 | 500 |
| Cray T3E | NUMA | 3维环网 | 2048 | 300 |
| HP V series | SMP | 8×8交叉开关 | 32 | 1000 |
| HP AlphaServer GS | SMP | 开关总线 | 32 | 400 |

。

## 大规模并行处理机

8.6.1 并行计算机系统结构

目前流行的高性能并行计算机系统结构通常可以分成以下5类：

并行向量处理机（PVP）

对称式共享存储器多处理机（SMP）

分布式共享存储器多处理机（DSM）

大规模并行处理机（MPP）

机群计算机（Cluster）



并行向量处理机

Cray C-90和 Cray T-90是这类机器的代表。

PVP系统一般由若干台高性能向量处理机（VP）构成。这些向量处理机是专门设计和定制的，拥有很高的向量处理性能。

PVP中经常采用专门设计的高带宽的交叉开关网络，把各VP与共享存储器模块SM连接起来。

这样的机器通常不使用Cache，而是使用大量的向量寄存器和指令缓冲器。

对称式共享存储器多处理机和分布式共享存储器多处理机

大规模并行处理机

Intel Paragon和IBM SP2是这类机器的代表。

MPP往往是超大规模的计算机系统。

具有以下特点：

处理结点使用商用微处理器，而且每个结点可以有多个微处理器；

具有较好的可扩放性，能扩展成具有成百上千个处理器；

系统中采用分布非共享的存储器，各结点有自己的地址空间；

采用专门设计和定制的高性能互连网络；

采用消息传递的通讯机制。

机群计算机

一种价格低廉、易于构建、可扩放性极强的并行计算机系统。它由多台同构或异构的独立计算机通过高性能网络或局域网互连在一起，协同完成特定的并行计算任务。

Berkeley NOW和SP2是这类机器的代表。

机群的主要特点

每个结点都是一台完整的计算机，拥有本地磁盘和操作系统，可以作为一个单独的计算资源供用户使用。

机群的各个结点一般通过商品化网络连接在一起；

网络接口以松散耦合的方式连接到结点的I/O总线

大规模并行处理机

MPP的出现和发展

从20世纪80年代末开始，MPP系统逐渐地显示出代替和超越向量计算多处理机系统的趋势。

早期的MPP：Intel Paragon(1992年)、KSR1.Cray T3D(1993年)、IBM SP2(1994年)等。

（ 分布存储的MIMD计算机）

MPP的高端机器：1996年Intel公司的ASCI Red

1997年SGI Cray公司的T3E900

（万亿次浮点运算的高性能并行计算机）

在这个时期，消息传递的大规模并行处理系统得到了迅速发展。

从90年代后期开始，基于消息传递的MPP系统慢慢地从主流的并行处理市场退出。

随着网络技术的发展，机群系统和MPP系统的界限越来越模糊。

90年代后期以来高性能计算机系统结构发展的一个趋势，新涌现的高性能计算机系统大多数都是由可扩放的高速互连网络连接的基于RISC微处理器的对称多处理机机群。

机群系统已经成了构建超大规模并行计算机系统的主要模式，MPP则是在慢慢地退居二三线了。

MPP系统概述

MPP结构的一个重要特性：可扩放性

处理器的数量可扩放至数千个处理器。

主存、I/O能力和带宽也能随处理器数量的增长而成比例地增长。

MPP主要采用了以下技术来提高系统的可扩放性。

使用物理上分布的主存体系结构，使分布式主存的总容量和总带宽能随处理结点数量的增加而增加。

处理能力、主存与I/O能力平衡发展。

计算能力与并行性平衡发展。

|  |  |  |  |
| --- | --- | --- | --- |
| MPP模型 | Intel/Sandia ASCI  Option Red | IBM SP2 | SGI/Cray Origin 2000 |
| 典型配置 | 9072个处理器  1.8 Tflop/s（NSL） | 400个处理器  100 Gflop/s（MHPCC） | 128个处理器  51 Gflop/s（NCSA） |
| 推出日期 | 1996年12月 | 1994年9月 | 1996年10月 |
| CPU类型 | 200 MHz，200 Mflop/s  Pentium Pro | 67 MHz，267 Mflop/s  POWER2 | 200 MHz，400Mflop/s  MIPS R10000 |
| 结点结构  数据存储 | 2个处理器，  32～256MB主存，  共享磁盘 | 1个处理器，  64MB～2GB本地主存，  1GB～14.5G本地磁盘 | 2个处理器，  64MB～256MB分布共享  主存和共享磁盘 |
| 互连网络 | 分离二维网孔 | 多级网络 | 胖超立方体网格 |
| 访存模型 | NORMA | NORMA | CC-NUMA |
| 结点OS | 轻量级内核（LWK） | 完全AIX（IBM UNIX） | 微内核Cellular IRIX |
| 编程语言 | 基于PUMA Portals  的MPI | MPI和PVM | Power C,  Power Fortran |
| 其他编程模型 | NX, PVM, HPF | HPF, Linda | MPI, PVM |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 结构特性 | IBM SP2 | | Cray T3D | | Cray T3E | | Intel Paragon | | | Intel/Sandia  Option Red |
| 典型配置 | 400个结点  100 Gflop/s | | 512个结点  153 Gflop/s | | 512个结点  1.2 Tflop/s | | 400个结点  40 Gflop/s | | | 4536个结点  1.8 Tflop/s |
| 推出日期 | 1994年 | | 1993年 | | 1996年 | | 1992年 | | | 1996年 |
| CPU类型 | 67 MHz  267 Mflop/s  POWER2 | | 150 MHz  150 Mflop/s  Alpha 21064 | | 300 MHz  600 Mflop/s  Alpha 21164 | | 50 MHz  100 Mflop/s  Intel i860 | | | 200 MHz  200 Mflop/s  Pentium Pro |
| 结点结构  数据存储 | 1CPU  64MB～2GB  本地存储器，  1～4.5GB  本地磁盘 | | 2CPU  64MB主存，  50GB共享磁盘 | | 4～8CPU  256MB～16GB  DSM 主存，  共享磁盘 | | 1～2CPU  16～128MB  本地存储器，  40GB共享磁盘 | | | 2CPU  32～256MB  本地存储器，  共享磁盘 |
| 互连网络 | 多级网络 | | 三维环绕 | | 三维环绕 | | 二维网孔 | | | 分离二维网孔 |
| 访存模型 | NORMA | | NUMA | | NCC—NUMA | | NORMA | | | NORMA |
| 结构特性 | | IBM SP2 | | Cray T3D | | Cray T3E | | Intel Paragon | Intel/Sandia  Option Red | | |
| 结构特性 | | IBM SP2 | | Cray T3D | | Cray T3E | | Intel Paragon | Intel/Sandia  Option Red | | |
| 结点OS | | 完全 AIX  （IBM UNIX） | | 微内核 | | 基于Chorus的  微内核 | | 微内核 | 轻量级内核  （LWK） | | |
| 编程模型 | | 消息传递 | | 共享变量、  消息传递、  PVM | | 共享变量、  消息传递、  PVM | | 消息传递 | 基于PUMA  Portals消息传递 | | |
| 编程语言 | | MPI、PVM、  HPF、Linda | | MPI、HPF | | MPI、HPF | | NX、MPI、  PVM | NX、PVM、  HPF | | |
| 点到点  通信延迟 | | 40μs | | 2μs | | N/A | | 30μs | 10μs | | |
| 点到点带宽 | | 35 MB/s | | 150 MB/s | | 480 MB/s | | 175 MB/s | 380 MB/s | | |

多处理机实例1：T1

SUN公司于2005年作为服务器处理器发布的多核多处理器。

同时采用了多线程和多核技术，全面提高吞吐率。

每个T1处理器中有8个处理器核，每个核最多支持4个线程。

每个处理器核中都有一条6段的单流出流水线。

（类似于前面介绍的5段流水线，只是增加了一个段，用于进行线程切换。 )

采用细粒度多线程，在每个时钟周期都可以切换到新的线程，而且在调度时可以跳过那些因流水线延迟或Cache不命中而处于等待状态的线程。

T1处理器只有在所有其4个线程都处于等待或停顿时才会出现空闲状态。

load和分支指令会导致3个时钟周期的延迟，不过它们都可以通过执行其他线程而被隐藏。

T1的组成结构

有8个核，每个核中都带有一个第一级Cache。

8个核通过一个交叉开关与4个第二级（L2）Cache相连。

用目录表法来实现Cache内容的一致性 。



|  |  |
| --- | --- |
| 特征 | Sun T1 |
| 多处理器和  多线程支持 | 每芯片8个核，每核4个线程。细粒度线程调度。  8个核共享一个浮点运算部件。支持片内多处理器。 |
| 流水线结构 | 简单的按序6段流水线，load和分支的延迟为3个时钟周期。 |
| 一级Cache | 16KB指令Cache，8KB数据Cache。64字节块大小。  在无竞争的情况下，L1不命中的开销是23个时钟周期。 |
| 二级Cache | 4个独立的二级Cache，每个750KB且和存储体相连。64字节块大小。  在无竞争的情况下，L2不命中的开销是110个时钟周期。 |
| 初始版本 | 90nm工艺，最高时钟频率1.2GHz，电源功率79W，  300M个晶体管，圆片面积大小379mm2。 |

T1的性能

基准测试程序：TPC-C、SPECJBB、SPECWeb99

在以下不同情况下L2 Cache的不命中率

容量分别为：1.5MB、3MB和6MB

块大小分别：32B和64B



在不同容量和块大小的情况下（与上图同），L2 Cache的不命中延迟



T1的每线程CPI、每核CPI以及有效的IPC（每个时钟周期完成的指令数）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 基准测试程序 | 每线程CPI | 每核CPI | 8个核的  有效CPI | 8个核的  有效IPC |
| TPC-C | 7.2 | 1.8 | 0.225 | 4.4 |
| SPECJBB | 5.6 | 1.40 | 0.175 | 5.7 |
| SPECWeb99 | 6.6 | 1.65 | 0.206 | 4.8 |

有效IPC=8÷每核CPI

4种多核处理器的性能对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特征 | SUN T1 | AMD Opteron | Intel Pentium D | IBM Power 5 |
| 核 | 8 | 2 | 2 | 2 |
| 每个核每时钟  周期发射的指令 | 1 | 3 | 3 | 4 |
| 多线程 | Fine-grained | No | SMT | SMT |
| Cache | 16/8 | 64/64 | 12k uops/16 | 64/32 |
| 一级  I/D in KB per core | 3MB shared | 1MB/core | 1MB/core | 二级：  1.9MB shared |
| 二级  Per core/shared |  |  |  | 三级：36MB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特征 | SUN T1 | AMD Opteron | Intel Pentium D | IBM Power 5 |
| 三级（off-chip） |  |  |  |  |
| 存储器带宽峰值  （DDR2 DRAMS） | 34.4GB/s | 8.6GB/s | 4.3GB/s | 17.2GB/s |
| MIPS峰值 | 9600 | 7200 | 9600 | 7600 |
| FLOPS | 1200 | 4800(w.SSE) | 6400(w.SSE) | 7600 |
| 时钟频率（GHz） | 1.2 | 2.4 | 3.2 | 1.9 |
| 晶体管数量（百万） | 300 | 233 | 230 | 276 |
| 晶片面积（mm2） | 379 | 199 | 206 | 389 |
| 电源功率（W） | 79 | 110 | 130 | 125 |

除了是重点开发ILP还是TLP的区别外，这些多核处理器还有一些根本的不同。

它们在对浮点运算提供的支持以及浮点运算的性能上有很大的不同。

它们的多处理器扩展能力不同，这对存储器的设计以及外部接口的使用有很大的影响。

Power5的可扩展性是最好的

所用的实现技术差别很大，难以对它们的晶片大小和功耗进行比较。

对存储器系统及其带宽的要求不同。

4种多核处理器的性能

以SPECRate、SPECJBB2005、SPECWeb05以及类TPC-C测试基准程序为负载

图中所有的数据都对Pentium D的数据进行了归一化处理，即Pentium D的值都是1。



多处理机实例2：Origin 2000

Origin 2000系列可扩展服务器产品

该系列包括：Origin 200、Origin 2000 Deskside、Origin 2000 Rack和Cray Origin 2000 4种机器。

Origin 2000 Deskside桌面服务器系统支持的处理器数目最多为8个

Origin 2000 Rack机柜服务器系统支持的处理器数目最多为16个

Cray Origin 2000服务器系统具有大规模扩充能力，支持的处理器数目最多可达到128个。

Origin 2000系列服务器产品优点

不仅具有SMP的易编程和平稳扩充特性，而且还具有MPP的高可扩放性，应用非常广泛。

该系列服务器综合平衡了高性能、可扩放性、可用性和兼容性，能满足许多应用的需求。

Origin 2000 服务器系列的I/O带宽可达102CB/s，系统传输速率比同类SMP服务器快几十倍。

（处理、存储和传输各种多媒体信息的理想系统）

Origin 2000的关键技术

CrayLink开关网络技术

多重交叉开关互连技术，用于连接处理器、存储器、I/O设备等。

替代总线成为处理器结点之间的互连网络。

使Origin 2000 系统成为模块化系统，系统规模可以是一个基本的模块，也可以是若干模块的互连，而且还可以方便地通过增加模块数量来扩充。

Origin 2000 系统的可扩放性体系结构最多可以扩展至1024个处理器，而且规模增加可使系统性能也呈线性增长，包括计算能力、主存容量和带宽、系统互连带宽、I/O带宽和网络连接能力。

通过CrayLink，分布在所有处理器结点上的存储器在逻辑上形成单一寻址空间的共享存储器系统，但对本地和远程存储器访问的时间是不同的，是一个NUMA结构。

Cellular IRIX操作系统

工业界最早投入使用的蜂窝式操作系统。

将操作系统功能分布到各个处理器结点上，可以实现从小系统到大系统的无缝扩展。

把多个相同的操作系统核心功能分别放到多个“蜂窝”（操作系统单元）中，每个蜂窝分别管理服务器中所有处理器的一个子集。每个操作系统单元都可以非常有效地扩展，单元之间互相通信，为用户提供一种单一的操作系统接口。

操作系统的这种蜂窝结构与积木式的硬件结构相结合，能够把故障隔离起来，可使故障局限于个别操作系统单元中，提高了服务器的可用性和可靠性。

从SGI的IRIX演变而来的，是以UNIX为基础的64位蜂窝式操作系统。

Origin 2000系列服务器的硬件结构：

结点板 （Origin200的主板 ）

组成部分

一个或两个MIPS R10000微处理器（内含第一级Cache）。其主频是180MHz或195MHz。

Origin 2000结点板结构



与处理器相配的第二级Cache，其容量为1MB或4MB；

主存储器（本地）以及用于实现Cache一致性的目录存储器；

用于实现互连的ASIC芯片，称为HUB。

提供了4个接口：

与处理器的接口

与存储器的接口

I/O接口

路由接口（接CrayLink互连网络）

HUB的结构

4个端口在内部以交叉开关互连，通过发送消息进行通信。

存储器接口能双向传送数据，最大传输率为780MB/s，



I/O和路由器接口各有两个半双工传送端口，最大传输率为2×780MB/s，即1.56GB/s。

每个Hub接口连接2个先进先出（FIFO）缓冲器，分别用于输入和输出的缓冲。

I/O子系统

由一组高速链路构成。称为Crosstalk （XTALK）。

Crosstalk I/O系统是分布的，在每个结点板上有一个I/O端口，可以被每个处理器访问。

I/O操作通过结点板上的单端口Crosstalk协议的链路进行控制，或者通过在Crossbow（XBOW）

ASIC芯片上的智能交叉开关进行互连。

XBOW ASIC芯片将Crosstalk I/O端口扩充到8个端口。

6个端口用于I/O

2个端口用于连接到结点板

互连网络子系统

互连网络子系统是由路由器和链路构成的。

每个路由器由一组交叉开关组成，能实现多路无阻塞连接。

每条双向链路带宽峰值达到1.6GB/s。

互连网络CrayLink Interconnect为每对结点提供至少两条独立链路进行通信。

这种结构使得结点之间的通信可以绕过不能运行的路由器和断开了的链路。

路由器将结点板上的HUB物理地连接到CrayLink Interconnect上。

路由器的核心：实现6路无阻塞交叉开关的路由ASIC芯片

路由器的交叉开关允许6个路由端口全双工同时操作，每个端口有2条单向的数据通路。

路由ASIC芯片的结构



ASIC芯片的主要功能

选择发送端口和接收端口的最高效连接，动态地切换6个端口的连接。

在CrayLink Interconnect的链路层协议（LLP）控制下与其他路由器和HUB进行可靠通信；

消息的包以虫蚀寻径方式通过路由器以减少通信时延；

对CrayLink信息提供缓存。

路由器提供的峰值通信带宽达到9.36GB/s。

不同的配置和互连

Origin 2000在不同处理器个数配置情况下的互连拓扑结构

处理器数目：4、16、32、64和128个

P：处理器

N：结点板

H：HUB

R：路由器

128处理器系统

由4个立方体组成，在立方体之间传送数据多经过了一级路由器。



在结点内部实现的是SMP（对称多处理器）结构，由于只有两个处理器，所以不存在SMP结构的总线瓶颈问题。

在结点之间实现的是大规模并行处理结构，但又解决了共享存储器问题。因此在Origin系统中，无论是访问存储器的时间还是结点间传送数据的带宽都很理想。

Origin系统中CPU访问存储器的延迟时间

假设：

CPU的主频为195MHz

Cache不命中

最小延迟时间：CPU访问本结点存储器的时间

最大延迟时间：CPU访问距离最远的存储器的时间

Origin系统中CPU访问存储器的延迟时间

|  |  |  |  |
| --- | --- | --- | --- |
| 系统CPU数 | 最小延迟时间 | 最大延迟时间 | 平均延迟时间 |
| 2 | 318ns | 343ns | 343ns |
| 4 | 318ns | 554ns | 441ns |
| 8 | 318ns | 759ns | 623ns |
| 16 | 318ns | 759ns | 691ns |
| 32 | 318ns | 836ns | 764ns |
| 64 | 318ns | 1067ns | 851ns |
| 128 | 318ns | 1169ns | 959ns |

Origin系统的带宽

每个Hub连到路由器和互连网络的最大频宽为：

1.56Gb/s（全双工，2×780Mb/s）

|  |  |  |
| --- | --- | --- |
| 系统处理器数 | 带宽  （无快速传送连线） | 带宽  （无快速传送连线） |
| 8 | 1.56Gb/s | 3.12Gb/s |
| 16 | 3.12Gb/s | 6.24Gb/s |
| 32 | 6.24Gb/s | 12.5Gb/s |
| 64 | 12.5Gb/s | -- |
| 128 | 25Gb/s | -- |

存储层次

寄存器、L1 Cache、L2 Cache和主存储器

寄存器和L1 Cache在R10000微处理器中

寄存器的存取时间最短

L1 Cache又分成指令Cache和数据Cache两部分

（避免取指令和存/取数据发生冲突）

L2 Cache安装在结点卡中，统一存放指令和数据，由SRAM组成。

主存储器地址是统一编址的，每个处理器通过互连网络可访问系统中任一存储单元。

实现Cache的一致性

基于目录协议与写作废协议

每个结点中，有一个存储器和一个目录存储器。

每块对应于一个目录项，每个目录项包含其对应存储器块的状态信息和系统中各Cache共享该存储块情况的位向量，根据位向量可以知道哪些Cache中有其副本。

## 机群系统

并行向量处理机（PVP）

对称多处理机（SMP）

大规模并行处理机（MPP）

分布共享存储多处理机（DSM）

机群（Cluster）

优势：低廉的价格、极强的灵活性和可扩缩性

成为近年来发展势头最为强劲的系统结构

基本结构

硬件组成

机群

一种价格低廉、易于构建、可扩缩性极强的并行计算机系统。

由多台同构或异构的独立计算机通过高性能网络或局域网互连在一起，协同完成特定的并行计算任务。

从用户的角度来看，机群就是一个单一、集中的计算资源。

一个简单PC机群的逻辑结构

4台PC机通过交换机连接在一起。

NIC表示网络接口，PCI表示I/O总线。

这是一种无共享的结构，大多数机群都采用这种结构。

如果将下图中的交换机换为共享磁盘，则可以得到共享磁盘的机群系统结构。

构成机群的每台计算机都被称为一个结点。

每个结点都是一个完整的系统，拥有本地磁盘和操作系统，可以作为一个单独的计算资源供用户使用。

除了PC机外，机群的结点还可以是工作站，甚至是规模较大的对称多处理机。

机群的各个结点一般通过商品化网络连接在一起。

网络接口与结点的I/O总线以松散耦合的方式相连。

机群的软件

机群操作系统：在各结点的操作系统之上建立一层操作系统来管理整个机群。

机群操作系统的功能

提供硬件管理、资源共享以及网络通信

实现单一系统映像

Single System Image，SSI

一项重要功能

机群的一个重要特征

SSI包含四重含义

单一系统

单一控制

逻辑上，最终用户或系统用户使用的服务都来自机群中唯一的位置。

系统管理员通过一个唯一的控制点配置机群的所有软、硬件组件。

对称性：用户可以从任一个结点上获得机群服务。

位置透明：用户不必了解真正提供服务的物理设备的具体位置。

简单机群系统中的SSI至少应该提供以下三种服务：

单一登录

即用户可以通过机群中的任何一个结点登录，而且在整个作业执行过程中只需登录一次，不必因作业被分派到其他结点上执行而重新登录。

单一文件系统

在机群系统中，有一些对整个机群所有结点而言都相同的软件，它们没有必要在每一个结点上重复安装。另外执行并行作业时要求每个结点都可以访问到可执行文件，即这些软件和可执行文件在整个机群系统中应该只有一个唯一的副本。

单一作业管理系统

用户可以透明地从任一结点提交作业，作业可以以批处理、交互或并行的方式被调度执行。

PBS、LSF、Condor和JOSS都是目前比较具有代表性的作业管理系统。

并行编程模型以及相关的并行编程环境

比较流行的并行编程工具：

MPI、PVM、OpenMP、HPF

机群系统的软件框架

机群系统的优点

系统开发周期短

结点采用商品化的PC机、工作站，通过商用网络连接。

系统开发的重点：通信子系统和并行编程环境

可靠性高

每个结点都是独立的PC机或工作站

可扩缩性强

机群的计算能力随着结点数量的增加而增大

机群结构灵活（结点之间以松耦合方式连接）

机群系统的硬件容易扩充和替换，可以灵活配置。

性能价格比高

用户编程方便

机群的迅猛发展还得益于微处理器技术、网络技术和并行程序设计技术的进步。

微处理器技术的进步使得微处理器的性能不断提高，价格不断下降 。

机群系统更容易融和到已有的网络系统中，而且随着网络技术的进步和高性能通信协议的引入，机群结点间的通信带宽进一步提高，通信延迟进一步缩短，逐步缓解了由于结点松散耦合引起的机群系统通信瓶颈问题。

随着PVM、MPI、HPF、OpenMP等并行编程模型的应用与成熟，使得在机群系统上开发并行应用更加方便。

机群的不足之处

由于机群由多台完整的计算机组成，它的维护相当于要同时去管理多个计算机系统，因此维护工作量较大，维护费用也较高。

机群的分类

根据组成机群的各个结点和网络是否相同，分为：

同构、异构

根据结点是PC还是工作站，分为：

PC机群、工作站机群

以机群系统的使用目的为依据，分为：

高可用性机群、负载均衡机群以及高性能机群

（最常用的分类方法 ）

高可用性机群

主要目的：当系统中某些结点出现故障的情况下，仍能继续对外提供服务。

采用冗余机制

当系统中某个结点由于软、硬件故障而失效时，该结点上的任务将在最短的时间内被迁移到机群内另一个具有相同功能与结构的结点上继续执行。

对于用户而言，系统可以一直为其提供服务。

适用于Web服务器、医学监测仪、银行POS系统等要求持续提供服务的应用。

负载均衡机群

主要目的：提供与结点个数成正比的负载能力

要求：机群能够根据系统中各个结点的负载情况实时地进行任务分配。

专门设置了一个重要的监控结点，负责监控其余每个工作结点的负载和状态，并根据监控结果将任务分派到不同的结点上。

适合大规模网络应用

如Web服务器或FTP服务器、大工作量的串行或批处理作业（如数据分析）

负载均衡机群适用于提供静态数据的服务；而高可用性机群既适用于提供静态数据的服务，又适用于提供动态数据的服务。

高性能计算机群

主要目的：降低高性能计算的成本

它通过高速的商用互连网络，将数十台乃至上千台PC机或工作站连接在一起，可以提供接近甚至超过传统并行计算机系统的计算能力，但其价格却仅是具有相同计算能力的传统并行计算机系统的几十分之一。

按照构建方式将机群分为：

（一种比较常用的分类方法）

专用机群

吞吐率较高，响应时间较短。

专用机群的结点往往是同构的，一般采用集中控制，由一个（或一组）管理员统一管理，而且用户一般需要通过一台终端机来访问它。

企业机群

各结点之间一般通过标准的LAN或WAN互连

通信开销较大、延迟较长

企业机群的各个结点一般是异构的

Berkeley NOW

美国加州大学Berkeley分校开发

具有很多优点：

采用商用千兆以太网和主动消息通信协议支持有效的通信。

通过用户级整合机群软件GLUNIX提供单一系统映像、资源管理和可用性，开发了一种新的无服务器网络文件系统xFS，以支持可扩缩性和单一文件层次的高可用性。

主动消息

实现低开销通信的一种异步通信机制

基本思想

在消息头部控制信息中携带一个用户级子例程（称作消息处理程序）的地址。

当消息头到达目的结点时，调用消息处理程序通过网络获取剩下的数据，并把它们集成到正在进行的计算中。

主动消息相当高效和灵活，以至于各种系统都逐渐地用它作为基本的通信机制。

GLUNIX

运行在工作站标准UNIX上的一个软件层，属于自包含软件。

主要思想

机群操作系统应由底层和高层组成。

其中底层是执行在核模式下的结点商用操作系统，高层是能提供机群所需的一些功能的用户级操作系统。

特别地，这一软件层能够提供机群内结点的单一系统映像，使得所有的处理器、存储器、网络容量和磁盘带宽均可以被分配给串行和并行应用无服务器文件系统xFS

一个无服务器的分布式文件系统。

将文件服务的功能分布到机群的所有结点上，以提供低延迟高带宽的文件系统服务功能。

主要采用廉价冗余磁盘阵列、协同文件缓存和分布式管理等技术。

Beowulf

目标

1G Flops的计算处理能力和10 GB的存储容量

价格不能过高

一个具有16个结点的机群

（Thomas Sterling与Don Becker二人构建）

硬件：Intel的DX4处理器以及10 Mbps的以太网

软件：基于Linux系统以及其他一些GNU软件

将这个系统命名为Beowulf

这种基于COTS（Commodity Off The Shelf）思想的技术也迅速由NASA传播到其他科研机构。

这类机群被称为Beowulf机群。

（Beowulf Class Cluster Computers）

Beowulf并不是一套具体的软件包或是一种新的网络拓扑结构，它只是一种思想。

在达到既定目标的前提下，把注意力集中在获取更高的性能价格比上。

9.4.3 LAMP

使用低成本、小配置（2～8个处理器）的SMP来构建机群系统逐渐成为主流。

这种结构的系统被统称为CLUMPs

（CLUster of MultiProcessors）

由于SMP结点内部与SMP结点之间通信能力往往不一致，CLUMPs一般使用专门的通信协议和通信算法。

LAMP（Local Area MultiProcessor）

由NEC实验室构建，基于Pentium Pro PC机、SMP机群

共有16个结点

每个结点包含两个Pentium Pro 200 MHz的CPU以及256 MB内存。

操作系统使用了支持SMP的Linux 2.0.34内核版本，提供MPICH 1.1.0并行程序开发环境。

同一个SMP结点内的两个CPU之间采用基于共享存储器的消息传递机制进行通信，而结点间通信则通过Myrinet完成。

IBM SP2

深蓝：采用30个RS/6000工作站（带有专门设计的480片国际象棋芯片）的IBM SP2机群

异步的MIMD，具有分布式存储器系统结构。

结点：一台RS/6000工作站，带有自己的存储器和本地磁盘。

结点中采用的处理器：一台6流出的超标量处理机

每个时钟周期可以执行6条指令，包括2条读数写数指令，2条浮点乘或加指令，1条变址增量指令和1条分支指令。

每个结点配有一套完整的AIX操作系统（IBM的UNIX）。

结点间的互连网络接口是松散耦合的，通过结点本身的I/O微通道（MCC）接到网络上，而不是通过本身的存储器总线。

SP2的结点数可以从2个到512个不等，除了每个结点采用RS/6000工作站外，整个SP2系统还需要配置另外一台RS/6000工作站作为系统控制台 。

SP2的结点可分为3类：宽结点、细结点、细2结点

它们都有：

1个指令Cache

1个数据Cache

1个分支指令和转移控制部件

2个整数部件

2个浮点部件

但它们在存储器容量、数据宽度和I/O总线插槽个数上有所不同。例如：

在存储器容量方面

宽结点：64～2048 MB

细结点和细2结点： 64～512 MB

在存储器总线的宽度方面

宽结点：256 b

细2结点：128 b

细结点：64 b

SP2的结点通过网络接口开关NIC接到HPS，IBM将其称为开关适配器。

SP2的I/O子系统的总体结构

SP2系统软件的核心：AIX操作系统

SP2中设置了一个专门的系统控制台用以管理整个系统，系统管理人员可以通过这个系统控制台从单一地点对整个系统进行管理。

## 多核系统

多核技术的好处

显著提升系统的计算能力，同时每个内核的主频可以比以前低系统的总体功耗增加不大

多核处理器采用了与单CPU相同的硬件体系结构，用户在提升计算能力的同时无需进行任何硬件上的改变。

现在，90%以上的个人计算机其处理器都是多核的。从2006年以来，在Intel和AMD两大处理器巨头的大力推动下，多核的普及已成为必然。

功耗与散热问题

随着芯片密度和时钟频率的不断提高，系统的功耗却呈现出指数性增长的趋势：

增加用户的使用成本

散热等诸多难题

摩尔定律的指引：

初期的几十MHz到近几年IBM的Power 6达到了4.75GHz；

2002年以来，CPU主频提升的困难越来越大；

从2006年开始，Intel和AMD都推出了多款面向服务器、工作站的多核处理器。

控制芯片密度也是一种有效的降低功耗的方法：

为了降低系统功耗，Cache占整个芯片面积的百分比比重越来越大。

并行度问题

处理器设计中，组织的变化主要集中在增加指令级级并行度上：

流水线技术、超标量技术、同时多线程技术等

缺点：

流水段越多，逻辑电路、互连结构以及控制信号就越复杂；

超标量组织也是通过增加并行流水线的个数来提高性能，需要更复杂的逻辑管理冲突和调度指令使用资源；

SMT技术中的线程在一组流水线上调度的复杂度也往往会限制线程的个数和可有效利用的流水线的个数，性能的改进也是有限的。

应用软件的问题

目前的绝大部分应用软件，特别是互联网应用软件都是面向多用户的多线程软件.

现在的数据库管理软件、数据库应用等服务器软件，一般要并行处理大量的、相对独立的事务。

多线程的本地应用、多进程应用

Java应用、多实例应用

超线程（Hyper Threading）和SMT技术只能在一定程度上支持多线程或多实例应用， 本质上还只是在一个执行核上运行。当线程个数较多时，就需要多核架构或并行处理机这样的处理系统了。

多核系统结构的需求（总结）

受CPU主频、功耗、散热和超标量等技术复杂度的限制，以及多线程应用软件需求的驱动，微处理器架构发展到多核成为一种必然的趋势.

多核架构也是摩尔定律驱动的结果，出现多核处理器最根本的原因是人们对计算能力永无止境的追求。

尽管这些年来，处理器从来没有停止过前进的脚步，但每一次性能的突破，换来的只是对更高性能的需求，特别是在油气勘探、气象预报、虚拟现实、人工智能等高度依赖于计算能力的场合，对性能的渴求更迫切

多核系统结构

多核技术是指在一枚处理器中集成两个或多个完整的计算内核，从而提高计算能力的技术。

按计算内核的对等与否，多核系统结构又可以分为同构多核结构和异构多核结构两种。

计算内核相同，地位对等的称为同构多核，反之称为异构多核。

需要注意的是，多核系统结构与多处理器不同，多处理器指多个CPU，每个CPU可以是单核或多核的。

多核的组织架构

多核处理器的组织架构主要包括：片上核心处理器的个数、多少级Cache、共享Cache的容量和内部互连结构等。

多核系统的4种典型的组织结构：

专用L1 Cache多核系统结构

专用L2 Cache多核系统结构

共享L2 Cache多核系统结构

共享L3 Cache多核系统结构

（A）专用L1 Cache多核系统结构

早期多核处理器的一种组织架构，现在在嵌入式芯片中仍能见到。在这种组织方式中，只有一级片内Cache，每个核带有自己的专用L1 Cache，分成指令Cache和数据Cache。这种组织的一个典型实例是ARM11 MPCore。

专用L2 Cache多核系统结构

专用L2 Cache多核系统结构无片内共享Cache，在这种结构里，片内有足够的可用面积容纳多个L2 Cache。这种组织的一个典型实例是AMD Opteron。

共享L2 Cache多核系统结构

共享L2 Cache多核系统结构采用了和专用L2 Cache多核结构类似的存储空间分配，不同的是该处理器架构拥有共享L2 Cache，Intel的Core Duo处理器就是这种结构

共享L3 Cache多核系统结构

共享L3 Cache多核系统结构出于性能上的考虑，分离出一个独立的三级Cache，每个CPU计算内核除了拥有专用的一、二级Cache外，还共享L3 Cache； Intel Core i7就是这种结构。

多核的组织架构

片内Cache是一种常见的技术和改善性能的方法。使用共享的片内L2 Cache 相对于专用Cache而言有如下几个优点：

共享片内L2 Cache可以减少整个系统的不命中概率；

多个核所共享的数据在共享Cache级上不需要复制；

局部线程能使用更多的Cache空间；

通过共享Cache能很容易地实现计算内核间的通信；

使用共享的L2 Cache将一致性问题限制在L1 Cache层次上，具有性能上的优点。

多核系统结构实例

多核CPU产品有很多，几乎所有的厂商都推出了自己的多核产品。本节介绍几个典型的多核系统结构实例：

Intel x86多核系统结构

Core Duo

Intel Core i7

面向嵌入式应用的ARM多核系统结构

Intel x86多核系统结构 - Core Duo

2006年推出的Core Duo是全球第一个低耗电的双核处理器（低于25瓦特）。

Core Duo实现了两个x86超标量处理器，共享二级Cache，Core Duo的每个核有自己的专用L1 Cache：一个32KB的指令Cache和一个32KB的数据Cache。

Intel x86多核系统结构 - Intel Core i7

i7是Intel于2008年11月推出的，实现了4个x86 SMT计算核，每个计算核带一个专用的L2 Cache、一个共享的L3 Cache。

在Core i7中，每个核拥有自己的专用L2 Cache，4个核共享一个8MB的L3 Cache。

ARM多核系统结构

ARM11 MPCore是基于ARM11处理器系列的多核产品，最多可配置4个处理器，每个处理器带有私有的L1指令Cache和L1数据Cache。

基于多核的并行程序设计

多核给我们提供了更经济的计算能力。但是，这种能力能否善加利用，还要取决于软件。

如果不针对多核进行软件开发，不仅多核提供的强大计算能力得不到利用，相反还有可能不如单核CPU好。

针对多核和多线程的软件开发将是未来十年软件开发的主要挑战，即基于多核的并行程序设计：

多核处理器的基本目的是通过多个任务的并行执行提高应用程序的性能；

尽量分解成多个独立任务，每个任务实现为一个线程，从而将多个任务分布到多个计算核上执行，减少程序的执行时间

并行编程模型

目前几种最重要的并行编程模型：

数据并行模型：编程级别比较高，编程相对简单，但它仅适用于数据并行问题；

消息传递模型：编程级别相对较低，但消息传递编程模型可以有更广泛的应用范围；

共享变量：采用多线程的方式，非常适合SMP共享内存多处理系统和多核处理器体系结构

数据并行和消息传递编程模式的对比

|  |  |  |
| --- | --- | --- |
| 对比内容 | 数据并行 | 消息传递 |
| 编程级别 | 高 | 低 |
| 适用的并行机类型 | SIMD/SPMD | SIMD/MIMD/SPMD/MPMD |
| 执行效率 | 效率依赖于编译器 | 高 |
| 地址空间 | 单一 | 多个 |
| 存储类型 | 共享内存 | 分布式或共享内存 |
| 通信的实现 | 编译器负责 | 程序员负责 |
| 问题类 | 数据并行类问题 | 数据并行任务并行 |
| 目前状况 | 缺乏高效的编译器支持 | 使用广泛 |

并行语言

并行程序是通过并行语言来表达的，并行语言的产生主要有三种方式：

设计全新的并行语言；

扩展原来的串行语言的语法成分使它支持并行特征；

不改变串行语言仅为串行语言提供可调用的并行库

并行算法

并行算法是给定并行模型的一种具体、明确的解决方法和步骤。

根据运算的基本对象的不同：

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

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）

根据并行计算任务的大小：

粗粒度并行算法（包含较长程序段和较大计算量）

细粒度并行算法（包含较短程序段和较小计算量）

介于二者之间的中粒度并行算法

从本质上说，不同的并行算法是根据问题类别的不同和并行机体系结构的特点产生出来的，一个好的并行算法要既能很好地匹配并行计算机硬件体系结构的特点，又能反映问题内在并行性。

多核编程实例

程序开发人员开发实际的并行程序主要方法是串行语言加并行库的扩展，其中比较典型的方法有两种：

共享存储的方法主要是采用多线程的方式，其主要程序开发环境就是已经成为事实工业标准的OpenMP，目前主要是商业编译器提供对该语言的支持；

消息传递开发则包括MPI和PVM等开源开发环境

本小结重点介绍基于OpenMP的多核编程环境

OpenMP（Open Multi-Processing）是一套支持跨平台共享内存方式的多线程并发的编程API，使用C，C++和Fortran语言，可以在大多数的处理器体系和操作系统中运行。

OpenMP采用可移植的、可扩展的模型，为程序员提供了一个简单而灵活的开发平台。

OpenMP提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的pragma来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。

一个简单的OpenMP程序：

并行执行后效率的提升：