第一章

1.2 为什么需要构建并行系统

• 单处理器性能大幅度提升的原因之一，是日益增加的集成电路晶体管密度。

一个简单的推导

• 更小的晶体管= 更快的处理器.

• 更快的处理器= 更高的能耗.

• 更高的能耗= 更高的热量.

• 更高的热量= 不稳定的处理器.

为什么需要编写并行程序

• 单核系统编写的程序无法利用多核处理器.

• 在多核系统中运行多个程序的实例

• 我们想要的是这个程序更快的运行

解决办法

• 将串行程序改成并行程序。

• 编写一个翻译程序来自动将串行程序翻译序成并行程序. （难）

怎样编写并行程序?

• 任务并行（Task parallelism ）

– 将待解决的问题所需要执行的各个任务分配到各个核上执行。

• 数据并行（Data parallelism ）

– 将待解决的问题所需要处理的数据分配给各个核。

– 每个核在分配到的数据集上执行大致相似的操作。

协调过程

• 通信（Communication ） – 一个或者多个核将自己的部分和结果发送到其他核。

• 负载平衡（Load balancing ） – 每个核分配大致相同数目的数据来计算。

• 同步（Synchronization ）– 在大多数系统

中，每个核有自己的执行空间，不能自动同步。

并行系统的类型

• 共享内存系统（Shared-memory ）

– 各个核能够共享访问计算机的内存。

– 通过检测和更新共享内存中的数据来协调各个核。

• 分布式内存系统（Distributed-memory ）

– 每个核都有拥有自己的私有内存。

– 核之间的通信是显式的，必须使用类似于在网络中发送消息的机制。

并发计算（Concurrent computing ）

– 一个程序的多个任务在同一个时间段内可以同时执行。

• 并行计算（Parallel computing ）

– 一个程序同一时刻通过多个任务紧密协作来解决某个问题。

• 分布式计算（Distributed computing ）

– 一个程序需要同一时刻与其他程序协作来解决某个问题。

第二章

主存（ （Main memory ）

• 主存中有许多区域，每个区域都可以存储指令和数据。

• 每个区域都有一个地址，可以通过这个地址来访问相应的区域以及区域中存储的数据和指令

中央处理单元 （CPU ）

• 分为两个部分：

• 控制单元（Control unit ）

– 负责决定应该执行程序中令的哪些指令 。 （ （the boss） ）

• 算数逻辑单元

（ Arithmetic and logic unit ，ALU ） ）

– 负责执行命令（ the worker）

关键术语

• 寄存器

– 快速存储介质，存储CPU 中的数据和程序执行的状态信息。

• 程序计数器

– 控制单元中一个特殊的寄存器，用来存放下一条指令的地址。

• 总线

– 在CPU 和主存之间进行数据和指令传输的互连结构。

进程、多任务及线程

操作系统

• 一种用来管理计算机的软件和硬件资源的主要软件。

• 一个进程包括：

– 可执行的机器语言程序

– 一个内存空间

– 资源描述符

– 安全信息

– 进程状态信息

多任务

操作系统提供对同时运行多个程序的支持

• 即使在单核系统中，通过时间片机制来实现多个进程的运行。

• 在一个程序执行了一个时间片的时间后，操作系统就阻塞当前程序，切换执行其他程序线程

• 一个进程包含一个或者多个线程。

• 将程序划分为多个大致独立的任务，当某个任务阻塞时能执行其他任务。

Cache基础知识

高速缓冲寄存器(CPU Cache) ，是一组相比于主存，CPU 能更快速地访问的内存区域。

CPU cache 位于与CPU 同一块的芯片或者位于其他比普通的内存芯片更快的访问的芯片上。

局部性原理：在一段时间内，CPU 总是几种地访问程序中的某一个部分而不是随机地对程序所有部分具有平均访问概率。（空间局部性 – 访问邻近的区域 时间局部性 – 在一段时间内）

Cache 映射

高速缓存行应该存储在什么位置？

• 全相联（Full associative ）

– 每个高速缓存行能够放置在Cache 中的任意位置。

• 直接映射（Direct mapped ）

– 放置在Cache 中指定的唯一位置。

• n 路组相联（n-way set associative ）

– 放置在Cache 中n

当内存中的行（多于一行）能被映射到Cache 中的多个不同位置时，替换Cache 中的哪一行？

最近最少使用

虚拟存储器

• 如果运行一个大型的程序，或者程序需要访问大型数据集，那么所有的指令或者数据可能在主存中放不下。

• 利用虚拟存储器，使得内存作为辅存的缓存。

– 它利用时间和空间局部性，通过在主存中存放当前执行程序所需要用到的部分。

交换空间（Swap space ）

– 那些暂时用不到的部分存储在辅存的块中

• 页（Pages ）

– 访问辅存比访问内存要慢几十万倍

– 页通常比较大

– 采用固定大小，从4~16K 。

虚拟页号

• 在编译程序时，给程序的页赋予虚拟页号。

• 当程序运行时，创建一张将虚拟页号应射程物理地址的表。

• 程序运行时使用到虚拟地址，页表将虚拟地址转换成物理地址。

转译后备缓冲区

• 高速联想存储器

• TLB 在快速存储介质中缓存了一些页表的条目，通常16-512 条。

• TLB 命中，TLB 缺失。

• 页面失效

– 假设想要访问的页不在内存中，即页表中该页没有合法的物理地址，该页只存储在磁盘上。

指令级并行

• 指令级并行，通过让多个处理器部件或者功能单元同时执行指令来提高处理器的性能。

• 实现方法有两种：

– 流水线

• 将功能单元分阶段安排

– 多发射

• 让多条指令同时启动

多发射

• 多发射处理器通过复制功能单元来同时执行程序中的不同指令。

• 如果功能单元是在编译时调度的，则称该多发射系统使用 静态多发射。

• 如果是在运行时调度的，则称该多发射系统使用动态多发射。 超标量superscalar

预测

• 为了能够利用多发射，系统必须找出能够同时执行的指令。

在预测技术中，编译器或者处理器对一条指令进行猜测，然后再猜测的基础上执行代码

硬件多线程

• 线程级并行（ Thread-Level Parallelism,TLP） ）

– 尝试通过 不同线程。 来提供并行性。

• 硬件多线程

– 为系统提供一种机制，使得当前执行的 任务被阻塞时，系统能够继续其他有用的工作。

• 细粒度多线程

– 处理器在 每条指令 执行完后切换线程，从而跳过被阻塞的线程。

– 缺点： 一个线程包括多条指令，而每条指令的切换都需要等待，导致延迟。

• 粗粒度多线程

– 只切换那些需要 等待较长时间才能完成操作的线程。

• 同步多线程（ Simultaneous multithreading， SMT ） ）

– 细粒度多线程的变种。

– 它通过允许多个线程同时使用多个功能单元来利用超标量处理器的性能。

网络性能指标

• 节点度（Node Degree ）射入或射出一个节点的边数。在单向网络中，入射和出射边之和称为节点度。

• 网络直径（Network Diameter ）： 网络中任何两个节点之间的最长距离，即最大路径数。

• 对剖宽度（Bisection Width ） ： 对分网络各半所必须移去的最少边数

• 对剖带宽（ Bisection Bandwidth ）: 每秒钟内，在最小的对剖平面上通过所有连线的最大信息位（或字节）数。

• 如果从任一节点观看网络都一样，则称网络为（ 对称的（Symmetry ）标 网络性能指标

• 用来衡量互联网络的性能指标:

• 延迟

– 指从发送源开始传送数据到目的地开始接收数间据之间的时间

• 带宽

– 指目的地在开始接收数据后接收数据的速度。

静态互连网络 与动态互连网络

• 静态互连网络：处理单元间有着固定连接的一类网络，在程序执行期间，这种点到点的链接保持不变典型的静态网络有一维线性阵列、二维网孔、树连接、超立方网络、立方环、洗牌交换网、蝶等 形网络等

• 动态网络：用交换开关构成的，可按应用程序的要求动态地改变连接组态；典型的动态网络包括总线、交叉开关和多级互连网络等。

静态互连网络

• 一维线性阵列（1-D Linear Array ）：

– 并行机中最简单、最基本的互连方式，

– 每个节点只与其左、右近邻相连，也叫二近邻连接，

– N 个节点用N-1 条边串接之，内节点度为2，为径为N-1， 为对剖宽度为1

– 当首、尾节点相连时可构成循环移位器，在拓扑结构上等同于环，环可以是单向的或双向的，其节点度恒为2 ，直径或为（双向为环）或为N-1（ (为单向环），对剖宽度为2

• （ 二维网孔（2-D Mesh ）：

– 每个节点只与其上、下、左、右的近邻相连（边界节点除外），节点度为4 ，网络直径为2（-1），对剖宽度为

– 在垂直方向上带环绕，水平方向呈蛇状，就变成Illiac 网孔了，节点度恒为4 ，网络直径为-1 ，对剖宽度为2

– 垂直和水平方向均带环绕，则变成了2-D 环绕（2-DTorus ）， 节点度恒为4 ，网络直径为2，对剖宽度为2

• 二叉树：

– 除了根、叶节点，每个内节点只与其父节点和两个子节点相连。

– 节点度为3 ，对剖宽度为1 ，而树的直径为

– 如果尽量增大节点度为，则直径缩小为2 ，此时就变成了星形网络，其对剖宽度为

– 传统二叉树的主要问题是根易成为通信瓶颈。胖树节点间的通路自叶向根逐渐变宽。

嵌入

 将去 网络中的各节点映射到另一个网络中去

 （ 用膨胀（Dilation ）系数来描述嵌入的质量，它是指被嵌入网络中的一条链路在所要嵌入的网络中对应所需的最大链路数

 如果该系数为1 ，则称为完美嵌入。

 到 环网可完美嵌入到2-D 环绕网中

 到 超立方网可完美嵌入到2-D

动态互连网络

• 总线：PCI 、VME 、Multics 、Sbus 、MicroChannel

– 多处理机总线系统的主要问题包括总线仲裁、中断处理、协议转换、快速同步、高速缓存一致性协议、分事务、总线桥和层次总线扩展等

• 总线的优点在于成本低，不随处理器数目的增加而增加

• 总线的缺点在于扩展性不好，总线的带宽固定，随着处理器数的增加，每个处理器带宽减少。

• 可利用程序中的局部性原理减少对总线带宽的需求

 （ 交叉开关（Crossbar ）网络是单级交换网络，可为每个端口提供更高的带宽。象电话交换机一样，交叉点开关可由程序控制动态设置其处于“开”或“关”状态，而能提供所有（源、目的）之间的动态连接。

 在并行处理中，交叉开关一般有两种使用方式：一种是用于对称的多处理机或多计算机机群中的处理器间的通信；另一种是用于SMP 服务器或向量超级计算机中处理器和存储器之间的存取。

• 千兆开关/FDDI: 一种用于构造Alpha 工作站和服务器互连的交叉开关 ，带宽3.6Gbps

• Sun Microsystem 公司在它们的Ultra Enterprisel0000 （StarFire ）SMP 服务器中，将Gigaplane 总线升级成Gigaplane-XB互联

• 处理器和存储器间的交叉开关 ：

– 交叉开关代替处理器和存储器间的连接总线

– 提供了多个处理器模块并行存取存储器的可能性

– 每个时刻每个存储器模块只能由一个处理器进行访问

• 交换开关模块：

– 一个交换开关模块有n 个输入和n 个输出，每个输入可连接到任意输出端口，但只允许一对一或一对多的映射，不允许多对一的映射，因为这将发生输出冲突

• 级间互连（Interstage Connection ）：

– 均匀洗牌、蝶网、多路均匀洗牌、交叉开关、立方连接

– n 输入的Ω 网络需要 级 开关，在Ilinois大学的Cedar[2] 多处理机系统中采用了Ω 网络

– Cray Y/MP 多级网络，该网络用来支持8 个向量处理器和256 个存储器模块之间的数据传输。网络能够避免8 个处理器同时进行存储器存取时的冲突。

标准互联网络

• Myrinet:

– Myrinet 是由Myricom 公司设计的千兆位包交换网络，其目的是为了构筑计算机机群，使系统互连成为一种商业产品。

– Myrinet 是基于加州理工学院开发的多计算机和VLSI 技术以及在南加州大学开发的ATOMIC/LAN 技术。Myrinet 能假设任意拓扑结构，不必限定为开关网孔或任何规则的结构。

– Myrinet 在数据链路层具有可变长的包格式，对每条链路施行流控制和错误控制，并使用切通选路法以及定制的可编程的主机接口。在物理层上，Myrinet 网使用全双工SAN 链路，最长可达3 米，峰值速率为（1.28 ＋1.28 ）Gbps（ （ 目前有2.56+2.56)

– Myrinet 交换开关 :8,12,16 端口

– Myrinet 主机接口 : 32 位的称作LANai 芯片的用户定制的VLSI处理器，它带有Myrinet 接口、包接口、DMA 引擎和快速静态随机存取存储器SRAM 。

– 140 of the November 2002 TOP500 use Myrinet, including15 of the top 100

• 高性能并行接口（HiPPI ）

– Los Alamos 国家实验室于1987 年提出的一个标准，其目的是试图统一来自不同产商生产的所有大型机和超级计算机的接口。在大型机和超级计算机工业界，HiPPI 作为短距离的系统到系统以及系统到外设连接的高速I/O 通道。

– 1993 年，ANSI X3T9.3 委员会认可了HiPPI 标准，它覆盖了物理和数据链路层，但在这两层之上的任何规定却取决于用户。

– HiPPI 是个单工的点到点的数据传输接口，其速率可达800Mbps 到1.6Gbps 。

– 开发成功了一种能提供潜在的6.4Gbps 速率，比HiPPI 快8 倍且有很低时延的超级HiPPI 技术，

– SGI 公司和Los Alamos 国家实验室都开发了用来构筑速率高达25.6Gbps 的HiPPI 交换开关的HiPPI 技术。

– HiPPI 通道和HiPPI 交换开关被用在SGI Power Challenge服务器、IBM 390 主机、Cray Y/MP 、C90 和T3D/T3E

• 光纤通道FC （Fiber Channel ） :

– 通道和网络标准的集成

– 光纤通道既可以是共享介质，也可以是一种交换技术

– 光纤通道操作速度范围可从100 到133 、200 、400 和800Mbps.FCSI 厂商也正在推出未来具有更高速度（1 、2 或4Gbps） ） 的光纤通道

– 光纤通道的价值已被现在的某些千兆位局域网所证实，这些局域网就是基于光纤通道技术的

– 连网拓扑结构的灵活性是光纤通道的主要财富，它支持点到点、仲裁环及交换光纤连

• FDDI :

– 光纤分布式数据接口FDDI （Fiber Distributed DataInterface ）

– FDDI 采用双向光纤令牌环可提供100-200Mbps 数据传输速率

– FDDI 具有互连大量设备的能力

– 传统的FDDI 仅以异步方式操作

• ATM （Asynchronous Transfer Mode ）:

– 由成立于1991 年的ATM 论坛和ITU 标准定义。

– ATM 是一种独立于介质的消息传输协议，它将消息段变成更短的固定长度为53 字节的报元进行传输。

– 这种技术是基于报元交换机制。ATM 的目的是将实时和突发数据的传输合并成单一的网络技术。

– ATM 网络支持从25 到51 、155 和622Mbps不同的速率，其速率越低ATM 交换器和使用的链路价格越低。

• SCI （Scalable Coherence Interface） ） 可扩展一致性接口

– 既能保持总线的优点又具有传统网络空间的可扩展性

– 通常的底板总线扩展成全双工、点到点的互连结构

– 提供分布共享存储器一致的高速缓存映象

– SCI 被设计用来提供低时延（小于1 微秒）和高带宽（高至8GB/s ）的点到点互连。

– 一旦SCI 得到充分开发，就可连接多至64K

并行计算机分类

• Flynn 分类（1966 年）

(1) 单指令流单数据流机SISD ，即传统的单处理机

(2) 单指令流多数据流机SIMD

(3) 多指令流单数据流机MISD ，实际中不存在的机器

(4) 多指令流多数据流机MIMD

• 并行机的结构模型－构 实际的机器体系结构

－SIMD (Single Instruction Multiple Data, 单指令流多数据流机)

－PVP (Parallel Vector Processor, 并行向量机)

－SMP (Symmetric Multiprocessor, 对称多处理机)

－MPP (Massively Parallel Processor, 大规模并行处理机)

－COW (Cluster of Workstation, 工作站机群)

－DSM (Distributed Shared Memory, 分布共享存储多处理机)

注：SIMD 是专用并行机，后5 种属于MIMD

单指令多数据流SIMD

• 通过对多个数据 执行相同的指令 从而实现在多个数据流上的操作。

– 一个控制单元和多个ALU

– 适合对大型数组的简单循环实行并行化

• 通过将数据分配给多个处理器 ，然后让各个处理器 使用相同的指令来操作数据子集实现并行化。这种并行也叫数据并行。

• 同步的。

SIMD 缺点

• 所有ALUs 都得执行同样的指令；

• ALU 必须同步操作；

• ALU 没有指令寄存器；

• 只适合解决大型数组的简单循环问题，而不能解决其他更复杂问题。

向量处理器Vector processors

• 能够对数组或者数组向量进行操作，而传统的CPU 是对单独的数据元素或者标量进行操作。

• 系统的特征： ：

• 向量寄存器

– 能够存储由多个操作数组成的向量，并能够同时对其内容进行操作。

• 向量化和流水化的功能单元

– 对向量中的每个元素做的是同样的操作。

• 向量指令

– 在向量上的操作的指令，而不是标量上的。

• 交叉存储器

– 内存系统可以看做由多个内存‘体’组成，每个内存提能够独立访问。

– 如果向量中的各个元素分布在不同的内存体中,那么在存入/ 存储连续数据时能够几乎无延迟的访问。

• 步长式存储器访问和硬件散射/ 聚集

– 程序能偶访问向量中固定间隔的元素

– 散射/ 聚集是对无规律间隔的数据进行读和写

向量处理器的优点

• 快

• 容易使用

• 向量编译器擅长识别向量化的代码

• 识别不能向量化的循环，并提供原因

• 很高的内存带宽

• 每个加载的数据都会被使用。

向量处理器缺点

• 它不能处理不规则的数据结构和其他的并行结构

• 处理更大的问题的能力有限，可扩展性差。

多指令多数据流系统MIMD

• 支持同时多个指令在多个数据流上操作。

• MIMD 系统通常包括一组完全独立的处理单元（核），每个处理单元都有自己的控制单元和ALU 。

• 异步的

共享内存系统 Shared Memory System

• 一组自治的处理器通过互连网络与内存系统相互连接。

• 每个处理器能够访问每个内存区域。

• 处理器之间通过访问共享的数据结构来进行隐式的通信。

共享存储的多处理机

• MIMD －多指令多数据流机

• 单一的共享地址空间

• 易于编程、难于扩展

• 存储访问可成为性能瓶颈

• 紧耦合与同构对称方式

优点

• 一个块在一个高速缓存中只缓存一个

– 所有被高速缓存的块只有一个拷贝

• 细粒度共享

– 通信延迟决定了存取路径适合的存储结构中的层次

• 2-10 时钟周期

• Cray Xmp 有共享的寄存器!

• 可能的正干扰

– 一个处理器从另一个处理器预取数据

• 缩小总存储量

– 两个处理器只用一个代码/ 数据拷贝

缺点

• 对高速缓存的带宽要求很高

• 增加了存取延迟

–X-bar

– 更大的cache

–L1 命中时间决定处理器周期 !!!

• 潜在的反相关

– 某个处理器需要另一个处理器的数据

消息传递与共享存储

• 集中共享存储系统中，所有处理器共享主存储器，统一编址，处理器之间的通信通过访问共享变量来实现。编程容易，不可扩放。

• 消息传递系统中，每个处理器都有一个只有它自己才能访问的局部存储器，单独编址，处理器之间的通信必须通过显式的消息传递来进行。扩放性好，编程困难。

分布式共享存储系统

• 共享存储器分布于各节点之中，节点之间通过可扩放性好的互连网络相连。

– 物理上分布存储的系统上逻辑地实现共享型存储模型

– 对于程序设计者隐藏了远程通信机制，保持了方便性和可移植性。

– DSM 系统底层分布式存储具有可扩放性和代价有效性

– 分布式的存储器和可扩放的互连网络增加了访存带宽，但却导致了不一致的访存结构

MPP

• 大规模并行处理机MPP(Massively Parallel Processor) 通常是指具有下列特点的大规模的计算机系统：

– 节点中使用商品化微处理器，且每个节点有一个或多个微处理器；

– 节点内使用物理上分布的存储器；

– 具有高通信带宽和低延迟的互连网络，节点间紧耦合；

– 能扩展成具有成百上千个处理器；

– 一个异步多指令流多数据流MIMD 机

• Intel Paragon 、IBM SP2 、Intel TFLOPS 和我国的曙光-1000等都是MPP

两种实现途径

– NCC-NUMA 体系结构，Cray T3E

– NORMA 体系结构，Intel/Sandia ASCI Option Red

• 与机群的概念很模糊

– 关键差别在于节点间的通信

– 差别缩小

新涌现的高性能计算系统绝大多数都将是由可扩放的高速互连机网络连接的基于商用微处理器的对称多处理机(SMP) 机群 ??

• 体积

• 功耗

• 散热

机群

• 机群是一组独立的计算机（节点）的集合体，通常有以下特征：

– 机群的各节点都是一个完整的系统：工作站，PC 机或SMP机器；

– 互连网络通常使用商品化网络，如以太网、FDDI 、ATM 等；

– 网络接口与节点的I/O 总线松耦合相连；

– 各节点通常有一个本地磁盘；

– 各节点有自己的完整的操作系统。

– 各节点除了可以作为一个单一的计算资源供交互式用户使用外还可以协同工作并表现为一个单一的、集中的计算资源供并行计算任务使用。

• 机群与分布式系统的区别：

– 机群继承了分布式系统的大部分知识

– 分布式系统通常是一个计算机的动物园，具有许多不同种类的计算机

– 机群通常是同构，耦合度较紧密，节点间互为信任关系

机群系统的迅速发展的原因

• 机群价格便宜并且易于构建； 穷人的解决方案，Gordon Bell 奖

• 作为机群节点的工作站系统的处理性能越来越强大；

• 局域网上新的网络技术和新的通信协议的引入，高带宽底延迟的节点间通信；

• 机群系统比传统的并行计算机更易于融合到已有的网络系统中去；

• 机群上的开发工具更成熟，传统并行计算机上缺乏一个统一的标准；

• 机群的可扩放性良好。

机群分类

•根据应用目标：

– 高性能机群（HP Cluster ）和高可用性机群（HA Cluster ）等；

• 根据节点的拥有情况：

– 专用机群（Dedicated Cluster ）：所有的资源是共享的，并行应用可以在整个机群上运行。

– 非专用机群（Nondedicated Cluster ）：全局应用通过窃取CPU时间获得运行，考虑进程迁移和负载平衡等问题。

• 根据节点的配置：

– 同构机群：各节点有相似的体系并且使用相同的操作系统。

– 异构机群，节点可以有不同的体系，运行的操作系统也可以不同

• 根据节点的硬件构成：

– 分为PC 机群CoPC （Cluster of PCs ）或称为PC 堆PoPC （Pile of PCs ），工作站机群COW （Cluster of Workstations ）和SMP 机群CLUMPs （Cluster of SMPs ）。

• 根据节点的操作系统：

– Linux 机群（如Beowulf ），Solaris 机群（如Berkeley NOW ），NT 机群（如HPVM ），AIX 机群（如IBM SP2 ）等。

并行计算机访存模型

• UMA （Uniform Memory Access） ） 模型是均匀存储访问模型的简称。其特点是：

– 物理存储器被所有处理器均匀共享；

– 所有处理器访问任何存储字取相同的时间；

– 每台处理器可带私有高速缓存；

– 外围设备也可以一定形式共享。

• NUMA(Nonuniform Memory Access) 模型是非均匀存储访问模型的简称.特点是：

– 被共享的存储器在物理上是分布在所有的处理器中的 ，其所有本地存储器的集合就组成了全局地址空间；

– 处理器访问存储器的时间是不一样的；访问本地存储器LM或群内共享存储器CSM 较快，而访问外地的存储器或全局共享存储器GSM 较慢( 此即非均匀存储访问名称的由来) ；

– 每台处理器照例可带私有高速缓存 ， 外设也可以某种形式共享。

• COMA(Cache-Only Memory Access) 模型是全高速缓存存储访问的简称 。 其特点是：

– 各处理器节点中没有存储层次结构 ， 全部高速缓存成了全局地址空间；

– 利用分布的高速缓存目录D 进行远程高速缓存的访问;

– COMA 中的高速缓存容量一般都大于2 级高速缓存容量；

– 使用COMA时数据开始时可任意分配 ， 因为在运行时它最终会被迁移到要用到它们的地方。

• 在多核系统中，各个核的Cache 存储相同变量的副本，当一个处理器更新Cache中该变量的副本时，其他处理器应该知道该变量已经更新，即其他处理器中Cache的副本也应该更新，这称为Cache 一致性问题。

• CC-NUMA （Coherent-Cache Nonuniform Memory Access） ） 模型是高速缓存一致性非均匀存储访问模型的简称。其特点是：

– 大多数使用基于目录的高速缓存一致性协议；

– 保留SMP 结构易于编程的优点，也改善常规SMP 的可扩放性；

– CC-NUMA 实际上是一个分布共享存储的DSM 多处理机系统；

– 它最显著的优点是程序员无需明确地在节点上分配数据，系统的硬件和软件开始时自动在各节点分配数据，在运行期间，高速缓存一致性硬件会自动地将数据迁移至要用到它的地方

NCC-NUMA（ （ 硬件不支持高速缓存一致性） ）

• 为了 避免一致性问题，共享数据被标识为不可高速缓存的，只有私有数据才能被高速缓存

• 好处在于仅需要很少的硬件支持就足够

• 缺点在于：

– ①支持透明的软件高速缓存一致性的编译机制非常有限，基于编译支持的软件高速缓存一致性是不太现实的。

– ②如果没有高速缓存一致性，那么在与访问远地单字所需的同等开销下系统将失去获取并使用一个高速缓存行中多个字的优点。当每次访问远地主存只能获得一个单字时，共享存储所具有的空间局部性的优点就荡然无存了。

– ③如果可以同时处理多个字（如一个高速缓存行）时，则诸如预取等延迟容忍技术效果才能更好。

 NORMA （No-Remote Memory Access） ） 模型是非远程

存储访问模型的简称。 。NORMA 的特点是：

 所有存储器是私有的；

 绝大数NUMA 都不支持远程存储器的访问；

 在DSM中，NORMA 就消失了。

共享虚拟存储SVM 结构

• SVM(Shared Virtual Memory) 系统 ，件 又称为软件DSM 系统，

– SVM 系统在基于消息传递的MPP 或机群系统中 ， 用软件把分布于节点的多个独立编址的存储器组织成一个统一编址的共享存储空间。

– 优点是在消息传递的系统上实现共享存储的编程界面 ， 但主要问题是难以获得满意的性能

• 与硬件共享存储系统相比，SVM 系统中较大的通信和共享粒度( 通常是存储页) 会导致假共享及额外的通信；

• 在基于机群的SVM 系统中 ，通信开销很大。基于SVM 系统的并行程序通信量通常比基于消息传递的并行程序的通信量大。

• SVM 系统的实现

– 在操作系统上改进 ，如 如Ivy 、Mermaid 、Mirage 和Clouds 等；

– 由运行系统来支撑 ，如 如CMU Midway 、Rice Munin 、Rice TreadMarks 、Utah Quarks 、DIKU CarlOS 、Maryland CVM和IAJIA 等；

– 从语言级来实现 ，如 如MIT CRL 、Linda 和Orca等 等。 。

– 混合实现的分布式共享存储系统 ， 其基本思想是结合软硬件实现的分布式共享存储系统的优点。

共享内存Shared Memory

• 动态线程Dynamic threads

– 主线程通常等待工作请求。当请求到达时，派生出一个工作线程来执行该请求；当线程工作完成时，就会终止执行再合并到主线程中去。

– 充分利用了系统的资源

– 但是线程的创建和终止消耗了时间

• 静态线程Static threads

– 主线程在完成必要的设置之后，派生出所有的线程，并且在工作结束前所有的线程都在运行

– 更好的执行，但是浪费系统资源。

阿姆达尔定律

• 大致上，除非一个串行程序的执行几乎全部都并行化，否则，不论有多少可以利用的核，通过并行化所产生的加速比都会受限。

可扩展性

• 如果一个技术可以处理规模不断增加的问题，那么它就是可扩展的。

• 如果在增加进程/ 线程的个数时，可以维持固定的效率，却不增加问题的规模，那么程序称为强可

扩展的。

• 如果在增加进程/ 线程个数的同时，只有以相同倍率增加问题的规模才能使效率值保持不变，那么

程序称为弱可扩展的。

计时Taking Timings

• 什么样的时间?

– 程序从开始到结束的时间

– 感兴趣的一部分所花费的时间

– CPU 时间

– ‘墙上时钟’时间— 代码从开始执行到执行结束的总耗费时间

Foster 并行化方法

1. 划分Partitioning: 将要执行的指令和数据按照计算部分拆分成多个小任务，关键在于识别出可以并行执行的任务。

2. 通信： 确定上一步所识别出来的任务之间需要执行哪些通信

3. 凝聚或者聚合: 将第一步和第二步所确定的任务与通信结合成更大的任务。

4. 分配程 ：将上一步聚合的任务分配到进程/线程中

第三章

进程

• 进程是一个程序，同时包含它的执行环境（内存、寄存器、程序计数器等），是操作系统中独立存在的可执行的基本程序单位。 +

• 通俗理解：串行应用程序编译形成的可执行代码，分为“指令”和“数据”两个部分，并在程序执行时“独立地申请和占有”内存空间，且所有计算均局限于该内存空间。

单机内的多个进程

• 多个进程可以同时存在于单机内同一操作系统：由操作系统负责调度分时共享处（ 机资源（CPU 、内存、存储、外设等）。

• 进程间相互独立（内存空间不相交）：在操作系统调度下各自独立地运行，例如多个串行应用程序在同一台计算机中运行

什么是MPI?

 Massage Passing Interface: 是消息传递函数库的标准规范，由MPI 论坛开发，支持Fortran 和C 。

 言 一种新的库描述，不是一种语言

 共有上百个函数调用接口，在Fortran 和C 语言中可以直接对这些函数进行调用

 是一种标准或规范，而不是特指某一个对它的具体实现

 MPI 是一种消息传递编程模型，并成为这种编程模型的代表和事实上的标准

为什么要使用MPI?

 高可移植性

 MPI 已在PC 机、MS Windows 以及所有主要的Unix 工作站上和所有主流的并行机上得到实现

 使用MPI 作消息传递的C 或Fortran 并行程序可不加改变地在上述平台实现

MPI 基本调用函数

• 从理论上说，MPI 所有的通信功能可以用它的6 个基本的调用来实现:

MPI\_INIT: 启动MPI环境

MPI\_COMM\_SIZE: 确定进程数

MPI\_COMM\_RANK: 确定自己的进程标识符

MPI\_SEND: 发送一条消息

MPI\_RECV: 接收一条消息

MPI\_FINALIZE: 结束MPI环境

通信子

• 一组可以相互发送消息的进程集合.

• MPI\_Init 在用户启动程序时，定义由用户启动的所有进程所组成的通信子.

• 称为 MPI\_COMM\_WORLD.

• 术语

– Blocking( 阻塞) : 一个例程须等待操作完成才返回, 返回后用户可以重新使用调用中所占用的资源.

– Non-blocking( 非阻塞): 一个例程不必等待操作完成便可返回, 但这并不意味着所占用的资源可被重用.

– Local( 本地): 不依赖于其它进程.

– Non-local( 非本地): 依赖于其它进程

MPI 消息

• MPI 消息包括信封和数据两个部分，信封指出了发送或接收消息的对象及相关信息，而数据是本消息将要传递的内容

• 数据：< 起始地址、数据个数、数据类型>

• 信封：< 源/ 目的、标识、通信域>

• 由count 个类型为datatype 的连续数据空间组成, 起始地址为buf

• 不是以字节数, 而是以元素的个数指定消息的长度

• count 可以是零, 这种情况下消息的数据部分是空的

• MPI 基本数据类型相应于宿主语言的基本数据类型

什么是缓冲区?

1. 应用程序中说明的变量，在消息传递语句中又用作缓冲区的起始位置。

2. 也可表示由系统( 不同用户) 创建和管理的某一存储区域，在消息传递过程中用于暂存放消息，也被称为系统缓冲区。

3. 用户可设置一定大小的存储区域，用作中间缓冲区以保留出现在其应用程序中的任意消息。

为什么使用消息标签(Tag)

为了说明为什么要用标签, 我们先来看右面一段没有使用标签的代码:

这段代码打算传送A 的前32个字节进入X, 传送B 的前16个字节进入Y. 但是, 如果消息B 尽管后发送但先到达进程Q,就会被第一个recv() 接收在X中 ，使用标签可以避免这个错误

MPI 集合通信

• 集合通信 (collective communication)是 一个进程组中的所有进程都参加的全局通信操作。

• 按照通信方向的不同，集合通信可分为三种类型：

– 一对多 ：一个进程向其它所有的进程发送消息，这个负责发送消息的进程叫做Root 进程。

– 多对一 ：一个进程负责从其它所有的进程接收消息，这个接收的进程也叫做Root 进程。

– 多对多：每一个进程都向其它所有的进程发送或者接收消息。

集合通信与点对点通信的不同

• 在通信子中的所有进程都必须调用相同的集合通信函数。

• 每个进程传递给MPI 集合通信函数的参数必须是“相容的”。

• 参数output\_data\_p 只用在dest\_process上 。

– 然而，所有进程仍需要传递一个与output\_data\_p 相对应的实际参数，即使它的值只是NULL 。

• 点对点通信函数 是通过标签和通信子来匹配的。

• 集合通信函数 不使用标签，只通过通信子和调用的顺序来进行匹配

第五章

Fork-Join 执行模式

 在开始执行的时候在 ，只有主线程的运行线程存在

 主线程在运行过程中 ，当遇到需要进行并行计算（ 的时候，派生出（Fork ，创建新线程或者唤醒已

务 有线程）线程来执行并行任务

 在并行执行的时候作 ，主线程和派生线程共同工作

 在并行代码结束执行后 ，派生线程退出或者挂起，不再工作，控制流程回到单独的主线程中（Join ，即多线程的会和）。

当程序到达parallel 指令时，究竟会发生什么？

 在parallel 之前，程序只使用一个线程

 到达parallel 时，原来的线程继续执行，thread\_count-1 个线程被启动。

线程组

 主线程

 从线程

在OpenMP 中的作用域

 一个能够被线程组中的所有线程访问的变量拥有共享作用域。

 一个只能被单个线程访问的变量拥有私有作用域。

 在parallel 指令前已经被声明的变量，拥有共享作用域；而在parallel 块中声明的变量拥有私有作用域。

归约操作符

 （ 归约操作符（reduction operator） ） 是一个二元操作（例如：加减法） 。

 （ 归约（reduction） ） 就是将相同的归约操作符重复地应用到操作数序列来得到一个结果的。

 所有的操作的中间结果都存储在同一个变量里：（ 归约变量（the reduction variable）。

锁是由一个数据结构和定义在这个数据结构上的函数组成，这些函数使得程序员可以显示地强制对临界区进行互斥访问