并行计算

一、并行计算概述

1. 并行计算定义：

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程。为执行并行计算，计算资源应包括一台配有多处理机（并行处理）的计算机、一个与网络相连的计算机专有编号，或者两者结合使用。并行计算的主要目的是快速解决大型且复杂的计算问题。此外还包括：利用非本地资源，节约成本 － 使用多个“廉价”计算资源取代大型计算机，同时克服单个计算机上存在的存储器限制。

为利用并行计算，通常计算问题表现为以下特征：

（1）将工作分离成离散部分，有助于同时解决；

（2）随时并及时地执行多个程序指令；

（3）多计算资源下解决问题的耗时要少于单个计算资源下的耗时。

并行计算是相对于串行计算来说的，所谓并行计算分为时间上的并行和空间上的并行。时间上的并行就是指流水线技术，而空间上的并行则是指用多个处理器并发的执行计算。

2. 并行化方法

1) 域分解

首先，确定数据如何划分到各个处理器

然后，确定每个处理器所需要做的事情

示例：求数组中的最大值

2) 任务（功能）分解

首先，将任务划分到各个处理器

然后，确定各个处理器需要处理的数据

Example: Event-handler for GUI

二、并行计算硬件环境

1. 并行计算机系统结构

1) Flynn 分类

a. MIMD

多指令流多数据流(Multiple Instruction Stream Multiple Data Stream，简称 MIMD)，它使用多个控制器来异步的控制多个处理器，从而实现空间上的并行性。

对于大多数并行计算机而言，多个处理单元都是根据不同的控制流程执行不同的操作，处理不同的数据，因此，它们被称作是多指令流多数据流计算机

b. SIMD

单指令流多数据流(Single Instruction Multiple Data)能够复制多个操作数，并把它们打包在大型寄存器的一组指令集，以同步方式，在同一时间内执行同一条指令。

以加法指令为例，单指令单数据（SISD）的 CPU 对加法指令译码后，执行部件先访问内存，取得第一个操作数；之后再一次访问内存，取得第二个操作数；随后才能进行求和运算。而在 SIMD 型的 CPU 中，指令译码后几个执行部件同时访问内存，一次性获得所有操作数进行运算。这个特点使 SIMD 特别适合于多媒体应用等数据密集型运算。

2) 并行计算及结构模型

a. SMP

SMP (Symmetric Multiprocessor)

- 采用商品化的处理器，这些处理器通过总线或交叉开关连接到共享存储器。每个处理器可等同地访问共享存储器、I/O 设备和操作系统服务。
- 扩展性有限。

- 曙光 1 号，IBM RS/6000

b. Cluster

Cluster（集群）

- 分布式存储，MIMD，工作站+商用互连网络，每个节点是一个完整的计算机，有自己的磁盘和操作系统，而 MPP 中只有微内核
- 优点：
  - 投资风险小
  - 系统结构灵活
  - 性能/价格比高
  - 能充分利用分散的计算资源
  - 可扩展性好
- 问题
  - 通信性能
  - 并行编程环境
- IBM Cluster 1350/1600

3) 内存访问模式

a. UMA

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

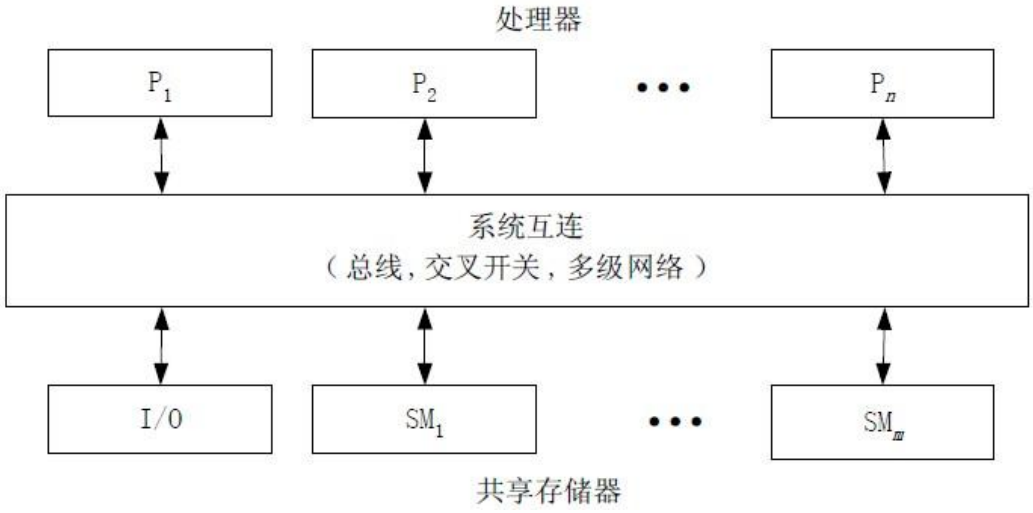
－物理存储器被所有处理器均匀共享；

－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。

- SMP 属于 UMA 型，单处理机（单地址空间，共享存储器）

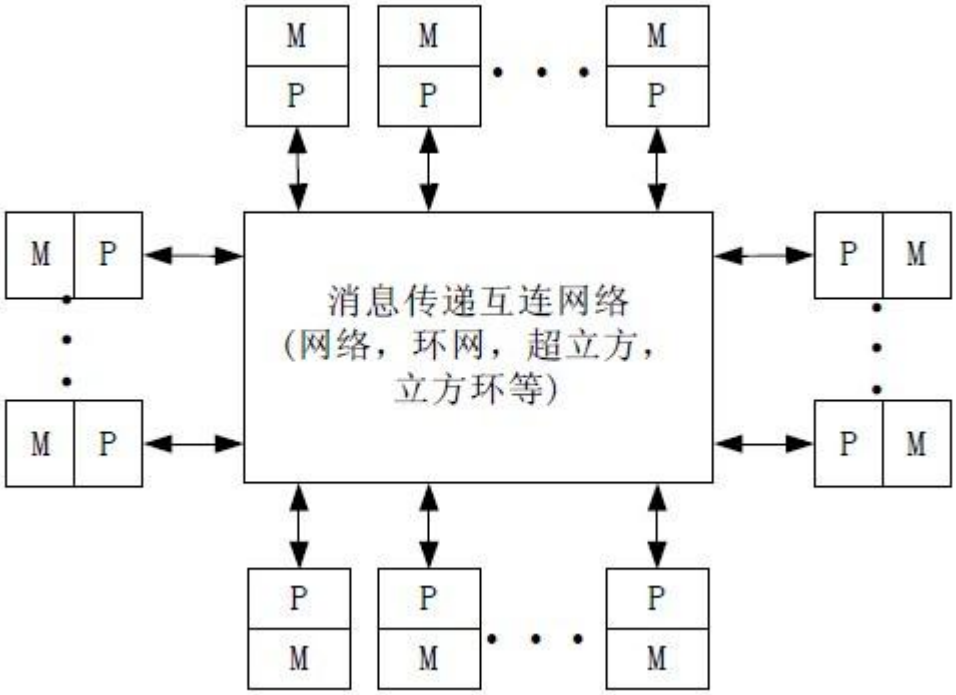


b. NORMA

- 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；

  －绝大多数 NUMA 都不支持远程存储器的访问。
- Cluster 属于 NORMA 型，多计算机（多地址空间非共享存储器）



2. 多核处理器

多核出现，原因有：

- 功耗过大，芯片温度过高
- 缓存过大，占据>70%芯片面积
- 隐式指令集并行，处理器指令执行流水化，加上不断提高的时钟速度，使得新一代的处理器芯片总能以更快的速度执行程序，而同时又保持顺序执行的假象。

➢ 多核时代

多核计算环境：

- 多个复杂度适中，相对低功耗的处理核心并行工作
- CPU 时钟频率基本不变
- 计算机硬件不会更快，但会更“宽”
- 操作系统、应用程序设计

三、内存系统，性能评测

1. 内存系统对性能的影响

➢ 对于很多应用而言，瓶颈在于内存系统，而不是 CPU

➢ 内存系统的性能包括两个方面：延迟和带宽

- 延迟：处理器向内存发起访问直至获取数据所需要的时间
- 带宽：内存系统向处理器传输数据的速率

2. 性能评测

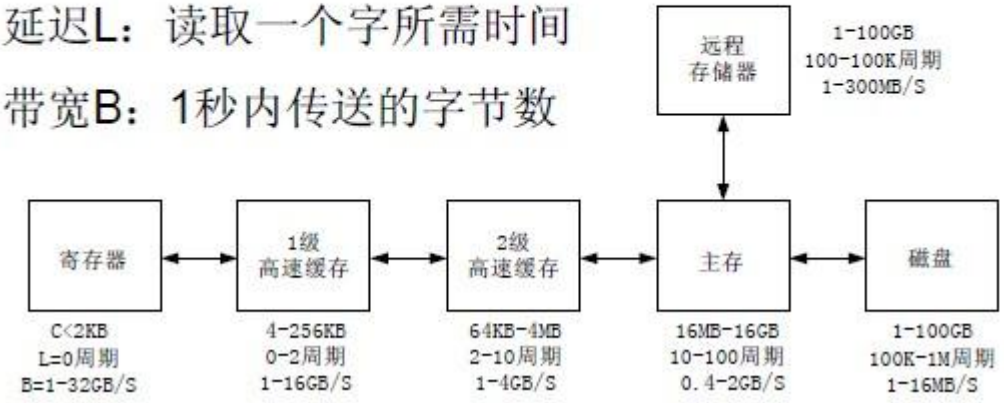
1). 基本性能指标

并行计算性能评测：存储器层次

容量C：能保存数据的字节数

延迟L：读取一个字所需时间

带宽B：1秒内传送的字节数



并行计算性能评测：CPU性能指标

- 工作负载
  - 执行时间 （Elapsed Time）
  - 浮点运算数（各种操作折算为浮点运算数的经验规则），Flop
  - 指令数目，MIPS
- 并行执行时间

T_{comput} 为计算时间，T_{paro} 为并行开销时间，T_{comm}为相互通信时间

$$T_n = T_{comput} + T_{paro} + T_{comm}$$

2). 加速比定律

- Amdahl 定律
  - 出发点：
    - －固定不变的计算负载；
    - －固定的计算负载分布在多个处理器上的，
    - －增加处理器加快执行速度，从而达到加速的目的。

Amdahl 定律：公式

- 固定负载的加速公式：

$$S = \frac{Ws + Wp}{Ws + W_P / p}$$

- $W_s + W_p$可相应地表示为f+（1-f）

$$S = \frac{f + (1-f)}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)}$$

- p→∞时，上式极限为：　　S= 1 / f

- Gustafson 定律
  - 出发点：
    - 对于很多大型计算，精度要求很高，即在此类应用中精度是个关键因素，而计算时间是固定不变的。此时为了提高精度，必须加大计算量，相应地亦必须增多处理器数才能维持时间不变；
    - －除非学术研究，在实际应用中没有必要固定工作负载而计算程序运行在不同数目的处理器上，增多处理器必须相应地增大问题规模才有实际意义。

Gustafson 定律：公式

- Gustafson加速定律：

$$S' = \frac{W_S + pWp}{W_S + p \cdot Wp / p} = \frac{W_S + pWp}{W_S + W_P}$$

$$S' = f + p(1\text{-}f) = p + f(1\text{-}p) = p\text{-}f(p\text{-}1)$$

- 并行开销W_o：

$$S' = \frac{W_S + pW_P}{W_S + W_P + W_O} = \frac{f + p(1-f)}{1 + W_O / W}$$

四、　多线程与 Pthread

1. 多线程基本概念

- 线程（thread）是进程上下文（context）中执行的代码序列，又被称为轻量级进程（light weight process）
- 在支持多线程的系统中，进程是资源分配的实体，而线程是被调度执行的基本单元。

➢ 线程与进程的区别

调度

- 在传统的操作系统中，CPU调度和分派的基本单位是进程。
- 在引入线程的操作系统中，则把线程作为CPU 调度和分派的基本单位，进程则作为资源拥有的基本单位，从而使传统进程的两个属性分开，线程便能轻装运行，这样可以显著地提高系统的并发性。
- 同一进程中线程的切换不会引起进程切换，从而避免了昂贵的系统调用。
  - 但是在由一个进程中的线程切换到另一进程中的线程时，依然会引起进程切换。

并发性

- 在引入线程的操作系统中，不仅进程之间可以并发执行，而且在一个进程中的多个线程之间也可以并发执行，因而使操作系统具有更好的并发性，从而能更有效地使用系统资源和提高系统的吞吐量。
  - 例如，在一个未引入线程的单CPU操作系统中，若仅设置一个文件服务进程，当它由于某种原因被封锁时，便没有其他的文件服务进程来提供服务。
- 在引入了线程的操作系统中，可以在一个文件服务进程中设置多个服务线程。
  - 当第一个线程等待时，文件服务进程中的第二个线程可以继续运行；当第二个线程封锁时，第三个线程可以继续执行，从而显著地提高了文件服务的质量以及系统的吞吐量。

拥有资源

- 进程
  - 不论是引入了线程的操作系统，还是传统的操作系统，进程都是拥有系统资源的一个独立单位，它可以拥有自己的资源。
- 线程
  - 线程自己不拥有系统资源（除部分必不可少的资源，如栈和寄存器），但它可以访问其隶属进程的资源。亦即一个进程的代码段、数据段以及系统资源（如已打开的文件、I/O设备等），可供同一进程的其他所有线程共享。

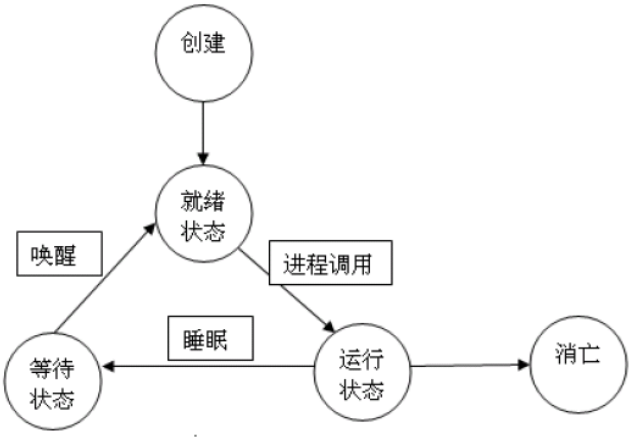
系统开销

- 进程
  - 创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等。
  - 在进行进程切换时，涉及到整个当前进程CPU 环境的保存环境的设置以及新被调度运行的进程的CPU 环境的设置。
- 线程
  - 切换只需保存和设置少量寄存器的内容，并不涉及存储器管理方面的操作。
  - 此外，由于同一进程中的多个线程具有相同的地址空间，致使它们之间的同步和通信的实现也变得比较容易。在有的系统中，线程的切换、同步和通信都无需操作系统内核的干预。

线程层次

- **用户级线程**在用户层通过线程库来实现。对它的创建、撤销和切换都不利用系统的调用。
- **核心级线程**由操作系统直接支持，即无论是在用户进程中的线程，还是系统进程中的线程，它们的创建、撤消和切换都由核心实现。
- **硬件线程**就是线程在硬件执行资源上的表现形式。
- 单个线程一般都包括上述三个层次的表现：用户级线程通过操作系统被作为核心级线程实现，再通过硬件相应的接口作为硬件线程来执行。

➢ 线程的生命周期



➢ 线程同步

竞争条件

- 在有些操作系统中，多个线程间可能会共享一些彼此都能够读写的公用存储区。
  - 可以在内存中，也可以是一个共享文件。
- 当两个或多个进程试图在同一时刻访问共享内存，或读写某些共享数据，而最后的结果取决于线程执行的顺序(线程运行时序)，就称为竞争条件(Race Conditions)

Bernstein 条件

$$I_1 \cap O_2 = \phi$$
$$I_2 \cap O_1 = \phi$$
$$O_1 \cap O_2 = \phi$$

I_i is the set of memory locations read by process P_i

O_j is the set of memory locations altered by process P_j

如果以上三个条件都得到了满足，则两个线程可以同步执行

临界区

- 临界区(critical section)是指包含有共享数据的一段代码，这些代码可能被多个线程执行。临界区的存在就是为了保证当有一个线程在临界区内执行的时候，不能有其他任何线程被允许在临界区执行。
- 设想有A,B两个线程执行同一段代码，则在任意时刻至多只能有一个线程在执行临界区内的代码。即，如果A线程正在临界区执行，B线程则只能在进入区等待。只有当A 线程执行完临界区的代码并退出临界区，原先处于等待状态的B线程才能继续向下执行并进入临界区。

信号量

- 信号量是E. W. Dijkstra 在1965 年提出的一种方法，可以用一个整数变量sem 来表示，对信号量有两个基本的原子操作：P（wait, 减量操作）和V（signal,增量操作）

多线程中的信号量

- 多线程环境中用来保证两个或多个关键代码段不被并发调用。
- 在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。
- 为了完成这个过程，需要创建一个信号量VI，然后将Acquire Semaphore VI以及Release Semaphore VI分别放置在每个关键代码段的首末端。确认这些信号量VI引用的是初始创建的信号量。
- Semaphore分为单值和多值两种，前者只能被一个线程获得，后者可以被若干个线程获得。

信号量的P、V操作

信号量S执行P操作：

```
Wait(semaphore S){
    S.value--;
    If( S.value <0){
        Add this process to S.L;
        Block();
}
}
```

信号量S执行V操作：

```
Signal(Semaphore S){
    S.value++;
    If(S.value < = 0){
        Remove a process P from S.L;
        Wakeup(P);
}
}
```

锁

- 锁 (Lock)类似于信号量，不同之处在于在同一时刻只能使用一个锁。锁对应的两个原子操作分别是:
  - Acquire(): 获取操作，将锁据为己有并把状态改为已加锁，如果该锁以被其他线程占有则等待锁状态变为未加锁状态。
  - Release(): 释放操作，将锁状态由已加锁改为未加锁状态。
- 一个锁最多只能由一个线程获得。任何线程对共享资源进行操作访问前必须先获得锁。否则，线程将保持在该锁地等待队列，直到该锁被释放。

互斥量（互斥锁）

- 互斥量（mutex 是 MUTual EXclusion 的缩写）是实现线程间同步的一种方法。
- 互斥量是一种锁，线程对共享资源进行访问之前必须先获得锁；否则线程将保持等待状态，直到该锁可用。只有其他线程都不占有它时一个线程才可以占有它，在该线程主动放弃它之前也没有另外的线程可以占有它。占有这个锁的过程就叫做锁定或者获得互斥量。

2. Pthread 多线程

五、 Java 多线程

见 PPT

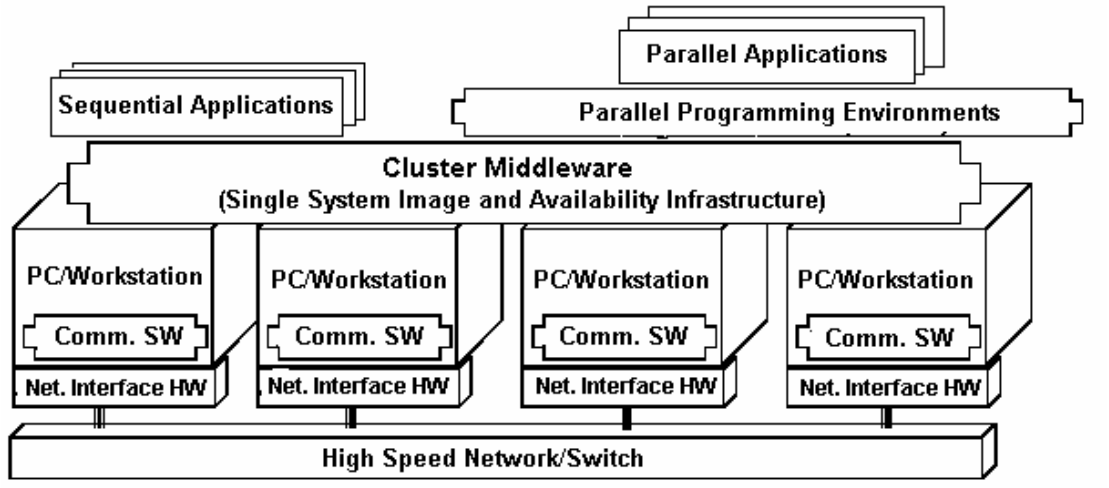
六、 OpenMP

见 PPT

七、 集群技术概述

1. 集群技术基础

- 定义
  - 集群概念最早由 IBM 于 20 世纪 60 年代提出 -Cluster（集群、机群、群集）
  - 集群一般由高速网络连接起来的高性能工作站或 PC 机组成。集群在工作中像一个统一的整合资源，所有节点使用单一界面。
- 体系结构



- 分类
  - 基于节点的所有者
    - 专用集群
    - 非专用集群
  - 基于节点的操作系统
    - Linux Clusters (Beowulf)
    - Solaris Clusters (Berkeley NOW)
    - AIX Clusters (IBM SP2)
    - SCO/Compaq Clusters (Unixware)
    - Windows Clusters
  - 基于节点的体系结构、配置和操作系统等
    - 同构集群
      - 所有节点配置相同
    - 异构集群
      - 不同节点有所差异
  - 用途
    - 高性能计算集群
      - 大规模科学计算
      - 海量数据存储与处理
    - 高可用集群
      - 负载均衡集群

八、 MPI

九、 MapReduce

十、 X10

十一、 并行程序设计方法学

- 并行算法设计
  - 并行算法的一般设计方法
    - 串行算法的直接并行化
      - 方法描述
        - 发掘和利用现有串行算法中的并行性，直接将串行算法改造为并行算法。

- 评注
  - 由串行算法直接并行化的方法是并行算法设计的最常用方法之一；
  - 不是所有的串行算法都可以直接并行化；
  - 一个好的串行算法未必能并行化为一个好的并行算法；
  - 许多数值串行算法可以并行化为有效的数值并行算法。
- 从问题描述开始设计并行算法
  - 方法描述
    - 从问题本身描述出发，不考虑相应的串行算法，设计一个全新的并行算法。
    - 评注
      - 挖掘问题的固有特性与并行的关系；
      - 设计全新的并行算法是一个挑战性和创造性的工作；
- 借用已有算法求解新问题
  - 方法描述
    - 找出求解问题和某个已解决问题之间的联系；
    - 改造或利用已知算法应用到求解问题上。
    - 评注
      - 这是一项创造性的工作；
      - 使用矩阵乘法算法求解所有点对间最短路径是一个很好的范例。

- PCAM 方法学、
  - 划分(Partitioning)
    - 分解成小的任务，开拓并发性；
  - 通讯(Communication)
    - 确定诸任务间的数据交换，监测划分的合理性；
  - 组合(Agglomeration)
    - 依据任务的局部性，组合成更大的任务；
  - 映射(Mapping)
    - 将每个任务分配到处理器上，提高算法的性能。
  - 小结

十二、 云计算

- GoogleDocs 是最早推出的云计算应用，是软件即服务思想的典型应用。它是类似于微软的 Office 的在线办公软件。它可以处理和搜索文档、表格、幻灯片，并可以通过网络和他人分享并设置共享权限。Google 文件是基于网络的文字处理和电子表格程序，可提高协作效率，多名用户可同时在线更改文件，并可以实时看到其他成员所作的编辑。用户只需一台接入互联网的计算机和可以使用 Google 文件的标准浏览器即可在线创建和管理、实时协作、权限管理、共享、搜索能力、修订历史记录功能，以及随时随地访问的特性，大大提高了文件操作的共享和协同能力。Google Docs 是云计算的一种重要应用，即可以通过浏览器的方式访问远端大规模的存储与计算服务。云计算能够为大规模的新一代网络应用打下良好的基础。