https://mp.weixin.qq.com/s/KbT7JKV-CS36L5oNAUULrA

**计算机系统漫游：贯穿计算机系统所有方面的重要概念**

计算机系统是由硬件和系统软件组成的，它们共同协作以运行应用程序。计算机内部的信息被表示为一组组的**位**，它们依据上下文有不同的解释方式。程序被其他程序翻译成不同的形式，开始时是**ASCII文本**，然后被编译器和链接器翻译成**二进制可执行文件**。

处理器读取并解释存放在主存里的二进制指令。因为计算机花费了大量的时间在内存、I/O设备和CPU寄存器之间复制数据，所以将系统中的存储设备划分成层次结构——**CPU寄存器在顶部**，接着是多层的**硬件高速缓存存储器**、**DRAM主存**和**磁盘存储器**。在层次模型中，位于更高层的存储设备比低层的存储设备要更快，单位比特造价也更高。层次结构中较高层次的存储设备可以作为较低层次设备的高速缓存。通过理解和运用这种存储层次结构的知识，程序员可以优化C程序的性能。

操作系统内核是应用程序和硬件之间的媒介。它提供三个基本的抽象：1）文件是对I/O设备的抽象；2）虚拟内存是对主存和磁盘的抽象；3）进程是处理器、主存和I/O设备的抽象。

最后，网络提供了计算机系统之间通信的手段。从特殊系统的角度来看，网络就是一种**I/O设备**。

**重要主题：**系统不仅仅只是硬件。系统是硬件和系统软件互相交织的集合体，它们必须共同协作以达到运行应用程序的最终目的。本书的余下部分会讲述硬件和软件的详细内容，通过了解这些详细内容，你可以写出更快速、更可靠和更安全的程序。

我们在此强调几个贯穿计算机系统所有方面的重要概念。我们会在本文多处讨论这些概念的重要性。

## Amdahl

Gene Amdahl，计算领域的早期先锋之一，对提升系统某一部分性能所带来的效果做出了简单却有见地的观察。这个观察被称为**Amdahl定律**（Amdahl’s law）。该定律的主要思想是，当我们对系统的某个部分加速时，其对系统整体性能的影响取决于该部分的重要性和加速程度。若系统执行某应用程序需要时间为Told。假设系统某部分所需执行时间与该时间的比例为α，而该部分性能提升比例为k。即该部分初始所需时间为αTold，现在所需时间为（αTold）/k。因此，总的执行时间应为

Tnew=（1-α）Told+(αTold)/k=Told［(1-α)+α/k］

由此，可以计算加速比S=Told/Tnew为:



举个例子，考虑这样一种情况，系统的某个部分初始耗时比例为60%（α=0.6），其加速比例因子为3（k=3）。则我们可以获得的加速比为1/［0.4+0.6/3］=1.67倍。虽然我们对系统的一个主要部分做出了重大改进，但是获得的系统加速比却明显小于这部分的加速比。这就是Amdahl定律的主要观点——**要想显著加速整个系统，必须提升全系统中相当大的部分的速度。**

Amdahl定律一个有趣的特殊情况是考虑k趋向于∞时的效果。这就意味着，我们可以取系统的某一部分将其加速到一个点，在这个点上，这部分花费的时间可以忽略不计。于是我们得到



举个例子，如果60%的系统能够加速到不花时间的程度，我们获得的净加速比将仍只有1/0.4=2.5×。

Amdahl定律描述了改善任何过程的一般原则。除了可以用在加速计算机系统方面之外，它还可以用在公司试图降低刀片制造成本，或学生想要提高自己的绩点平均值等方面。也许它在计算机世界里是最有意义的，在这里我们常常把性能提升2倍或更高的比例因子。这么高的比例因子只有通过优化系统的大部分组件才能获得。

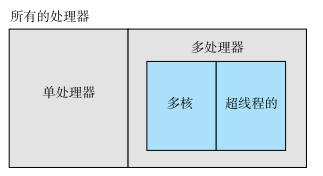
## 并行和并发

数字计算机的整个历史中，有两个需求是驱动进步的持续动力：一个是我们想要计算机做得更多，另一个是我们想要计算机运行得更快。当处理器能够同时做更多的事情时，这两个因素都会改进。我们用的术语**并发**（concurrency）是一个通用的概念，指一个同时具有多个活动的系统；而术语**并行**（parallelism）指的是用并发来使一个系统运行得更快。并行可以在计算机系统的多个抽象层次上运用。在此，我们按照系统层次结构中由高到低的顺序重点强调三个层次。

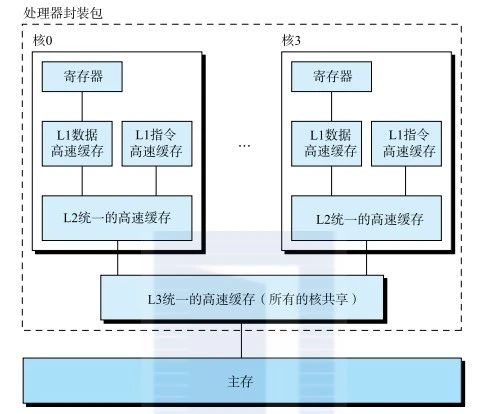
**1 线程级并发**

构建在进程这个抽象之上，我们能够设计出同时有多个程序执行的系统，这就导致了并发。使用线程，我们甚至能够在一个进程中执行多个控制流。自20世纪60年代初期出现时间共享以来，计算机系统中就开始有了对并发执行的支持。传统意义上，这种并发执行只是模拟出来的，是通过使一台计算机在它正在执行的进程间快速切换来实现的，就好像一个杂耍艺人保持多个球在空中飞舞一样。这种并发形式允许多个用户同时与系统交互，例如，当许多人想要从一个Web服务器获取页面时。它还允许一个用户同时从事多个任务，例如，在一个窗口中开启Web浏览器，在另一窗口中运行字处理器，同时又播放音乐。在以前，即使处理器必须在多个任务间切换，大多数实际的计算也都是由一个处理器来完成的。这种配置称为单处理器系统。

当构建一个由单操作系统内核控制的多处理器组成的系统时，我们就得到了一个多处理器系统。其实从20世纪80年代开始，在大规模的计算中就有了这种系统，但是直到最近，随着多核处理器和超线程（hyperthreading）的出现，这种系统才变得常见。图1给出了这些不同处理器类型的分类。



多核处理器是将多个CPU（称为“核”）集成到一个集成电路芯片上。图2描述的是一个典型多核处理器的组织结构，其中微处理器芯片有4个CPU核，每个核都有自己的L1和L2高速缓存，其中的L1高速缓存分为两个部分——**一个保存最近取到的指令，另一个存放数据**。这些核共享更高层次的高速缓存，以及到主存的接口。工业界的专家预言他们能够将几十个、最终会是上百个核做到一个芯片上。



**超线程**，有时称为同时多线程（simultaneous multi-threading），是一项允许一个CPU执行多个控制流的技术。它涉及CPU某些硬件有多个备份，比如程序计数器和寄存器文件，而其他的硬件部分只有一份，比如执行浮点算术运算的单元。常规的处理器需要大约20000个时钟周期做不同线程间的转换，而超线程的处理器可以在单个周期的基础上决定要执行哪一个线程。这使得CPU能够更好地利用它的处理资源。比如，假设一个线程必须等到某些数据被装载到高速缓存中，那CPU就可以继续去执行另一个线程。举例来说，Intel Core i7处理器可以让每个核执行两个线程，所以一个4核的系统实际上可以并行地执行8个线程。

多处理器的使用可以从两方面提高系统性能。首先，它减少了在执行多个任务时模拟并发的需要。正如前面提到的，即使是只有一个用户使用的个人计算机也需要并发地执行多个活动。其次，它可以使应用程序运行得更快，当然，这必须要求程序是以多线程方式来书写的，这些线程可以并行地高效执行。因此，虽然并发原理的形成和研究已经超过50年的时间了，但是多核和超线程系统的出现才极大地激发了一种愿望，即找到书写应用程序的方法利用硬件开发线程级并行性。

**2 指令级并行**

在较低的抽象层次上，现代处理器可以同时执行多条指令的属性称为指令级并行。早期的微处理器，如1978年的Intel 8086，需要多个（通常是3～10个）时钟周期来执行一条指令。最近的处理器可以保持每个时钟周期2～4条指令的执行速率。其实每条指令从开始到结束需要长得多的时间，大约20个或者更多周期，但是处理器使用了非常多的聪明技巧来同时处理多达100条指令。

如果处理器可以达到比一个周期一条指令更快的执行速率，就称之为**超标量**（superscalar）处理器。大多数现代处理器都支持超标量操作。

**3 单指令、多数据并行**

在最低层次上，许多现代处理器拥有特殊的硬件，允许一条指令产生多个可以并行执行的操作，这种方式称为**单指令**、**多数据**，即**SIMD并行**。例如，较新几代的Intel和AMD处理器都具有并行地对8对单精度浮点数（C数据类型float）做加法的指令。

提供这些SIMD指令多是为了提高处理影像、声音和视频数据应用的执行速度。虽然有些编译器会试图从C程序中自动抽取SIMD并行性，但是更可靠的方法是用编译器支持的特殊的向量数据类型来写程序，比如GCC就支持向量数据类型。