加和减少的潜在误解，当我们想说“改善性能”和“改善执行时间”的时候，通常说“增加性能”和“减少执行时间”。

1.6.2 性能的度量

时间是计算机性能的衡量标准：完成同样的计算任务，需要时间最少的计算机是最快的。程序的执行时间一般以秒为单位。然而，时间可以用不同的方式来定义，这取决于我们所计数的内容。对时间最直接的定义是挂钟时间（wall clock time），也叫响应时间（response time）、运行时间（elapsed time）等。这些术语均表示完成某项任务所需的总时间，包括了磁盘访问、内存访问、I/O 活动和操作系统开销等一切时间。

计算机经常被共享使用，一个处理器也可能同时运行多个程序。在这种情况下，系统可能更侧重于优化吞吐率，而不是致力于将单个程序的执行时间变得最短。因此，我们往往要把运行自己任务的时间与一般的运行时间区别开来。在这里可以使用 **CPU 执行时间**（CPU execution time）来进行区别，简称为 **CPU 时间**，只表示在 CPU 上花费的时间，而不包括等待 I/O 或运行其他程序的时间。（需要注意的是，用户所感受到的是程序的运行时间，而不是 CPU 时间。）CPU 时间还可进一步分为用于用户程序的时间和操作系统为用户程序执行相关任务所花去的 CPU 时间。前者称为**用户 CPU 时间**（user CPU time），后者称为**系统 CPU 时间**（system CPU time）。要精确区分这两种 CPU 时间是困难的，因为通常难以分清哪些操作系统的活动是属于哪个用户程序的，而且不同操作系统的功能也千差万别。

CPU 执行时间：简称为 CPU 时间，执行某一任务在 CPU 上所花费的时间。

用户 CPU 时间：程序本身所花费的 CPU 时间。

系统 CPU 时间：为执行程序而花费在操作系统上的时间。

为了一致，我们保持区分基于响应时间的性能和基于 CPU 执行时间的性能。我们使用术语**系统性能**（system performance）表示空载系统的响应时间，并用术语 **CPU 性能**（CPU performance）表示用户 CPU 时间。本章我们概括介绍计算机性能，虽然既适用于响应时间的度量，也适用于 CPU 时间的度量，但本章的重点将放在 CPU 性能上。

理解程序性能 不同的应用关注计算机系统性能的不同方面。许多应用，特别是那些运行在服务器上的应用，主要关注 I/O 性能，所以此类应用既依赖硬件又依赖软件，关注的是测量出的总执行时间。在其他一些应用中，用户可能对吞吐率、响应时间或两者的复杂组合更为关注（例如，最差响应时间下的最大吞吐率）。要改进程序的性能，必须明确定义性能指标，然后通过测量程序执行时间、查找可能的限制因素来找到性能瓶颈。在后续章节中，我们将介绍如何在系统的各个部分寻找瓶颈并改进性能。

虽然作为计算机用户我们关心的是时间，但当我们深入研究计算机的细节时，使用其他的度量可能更为方便。特别是对计算机设计者而言，他们需要考虑如何度量计算机硬件完成基本功能的速度。几乎所有计算机的构建都需要基于时钟，该时钟确定各类事件在硬件中何时发生。这些离散时间间隔被称为**时钟周期数**（clock cycle，或称**滴答数**、**时钟滴答数**、**时钟数**、**周期数**）。设计人员在提及**时钟周期**时，可能使用完整时钟周期的时间（例如 250 皮秒或 250ps），也可能使用时钟周期的倒数，即时钟频率（例如 4GHz），在下一小节中，我们将正式确定硬件设计者常用的时钟周期与计算机用户常用

时钟周期数：也叫滴答数、时钟滴答数、时钟数、周期数，为计算机一个时钟周期的时间，通常是指处理器时钟，并在固定频率下运行。

周期长度：每个时钟周期持续的时间长度。

的秒数之间的关系。

自我检测

1. 假设某应用同时使用了个人移动设备和云，其性能受网络性能限制。那么对于下列三种方法，哪种只改进了吞吐率？哪种同时改进了响应时间和吞吐率？哪种都没有改进？
   a. 在个人移动设备和云之间增加一条额外的网络信道，从而增加总的网络吞吐率，并减少网络访问的延迟（现在已经存在两条网络信道）。
   b. 改进网络软件，从而减少网络通信延迟，但并不增加吞吐率。
   c. 增加计算机的内存。
2. 计算机 C 的性能是计算机 B 的 4 倍，而计算机 B 运行给定的应用需要 28 秒，请问计算机 C 运行同样的应用需要多长时间？

1.6.3　CPU 性能及其度量因素

用户和设计者往往用不同的指标衡量性能。如果我们能够将这些不同的指标联系起来，就可以确定设计变更对用户可感知的性能的影响，由于我们都在关注 CPU 性能，因此性能度量的基本指标应该是 CPU 执行时间。一个简单公式可将最基本的指标（时钟周期数和时钟周期长度）与 CPU 时间联系起来：

$$\text{程序的 CPU 执行时间} = \text{程序的 CPU 时钟周期数} \times \text{时钟周期长度}$$

由于时钟频率和时钟周期长度互为倒数，故另一种表达形式为

$$\text{程序的CPU执行时间} = \frac{\text{程序的CPU时钟周期数}}{\text{时钟频率}}$$

这个公式清楚地表明，硬件设计者减少程序执行所需的 CPU 时钟周期数或缩短时钟周期长度，就能改进性能。在后面几章中我们将看到，设计者经常要面对这二者之间的权衡。许多技术在减少时钟周期数的同时也会增加时钟周期长度。

例题 | 改进性能

我们最喜欢的某个程序在时钟频率为 2GHz 的计算机 A 上运行需要 10 秒。现在尝试帮助计算机设计者建造一台计算机 B，将运行时间缩短为 6 秒。设计人员已经确定可以大幅提高时钟频率，但这可能会影响 CPU 其余部分的设计，使得计算机 B 运行该程序时需要相当于计算机 A 的 1.2 倍的时钟周期数。那么，我们应该建议计算机设计者将时钟频率设计目标确定为多少？

答案 | 我们首先要知道在 A 上运行该程序需要多少时钟周期数：

$$\text{CPU时间}_A = \frac{\text{CPU时钟周期数}_A}{\text{时钟频率}_A}$$

$$10\text{秒} = \frac{\text{CPU时钟周期数}_A}{2\times10^9 \frac{\text{时钟周期数}}{\text{秒}}}$$

$$\text{CPU时钟周期数}_A = 10\text{秒}\times2\times10^9 \frac{\text{时钟周期数}}{\text{秒}} = 20\times10^9 \text{时钟周期数}$$

在 B 上执行程序的 CPU 时间可用下述公式计算：

$$\text{CPU时间}_B=\frac{1.2\times\text{CPU时钟周期数}_A}{\text{时钟频率}_B}$$

$$6\text{秒}=\frac{1.2\times20\times10^9\text{时钟周期数}}{\text{时钟频率}_B}$$

$$\text{时钟频率}_B=\frac{1.2\times20\times10^9\text{时钟周期数}}{6\text{秒}}=\frac{0.2\times20\times10^9\text{时钟周期数}}{\text{秒}}$$

$$=\frac{4\times10^9\text{时钟周期数}}{\text{秒}}=4\text{GHz}$$

因此，要在 6 秒内运行完该程序，计算机 B 的时钟频率必须提高为 A 的 2 倍。

1.6.4 指令性能

上述性能公式并未涉及程序所需的指令数。然而，由于编译器明确生成了要执行的指令，且计算机必须通过执行指令来运行程序，因此执行时间必然依赖于程序中的指令数。一种考虑执行时间的方法是，执行时间等于执行的指令数乘以每条指令的平均时间。因此，一个程序需要的时钟周期数可写为：

$$\text{CPU 时钟周期数} = \text{程序的指令数} \times \text{指令平均时钟周期数}$$

指令平均时钟周期数（clock cycle per instruction）表示执行每条指令所需的时钟周期平均数，缩写为 CPI。根据所完成任务的不同，不同的指令需要的时间可能不同，CPI 是程序的所有指令所用时钟周期的平均数。CPI 提供了一种相同指令系统在不同实现下比较性能的方法，因为在指令系统不变的情况下，一个程序执行的指令数是不变的。

指令平均时钟周期数：表示执行某个程序或者程序片段时每条指令所需的时钟周期平均数。

例题 | 性能公式的使用

假设我们有相同指令系统的两种不同实现。计算机 A 的时钟周期长度为 250ps，对某程序的 CPI 为 2.0；计算机 B 的时钟周期长度为 500ps，对同样程序的 CPI 为 1.2。对于该程序，请问哪台计算机执行的速度更快？快多少？

答案 | 我们知道，对于固定的程序，每台计算机执行的总指令数是相同的，我们用 I 来表示该数值。首先，求每台计算机的 CPU 时钟周期数：

$$\text{CPU 时钟周期数}_A = I\times2.0$$
$$\text{CPU 时钟周期数}_B = I\times1.2$$

现在，可以计算每台计算机的 CPU 时间：

$$\text{CPU 时间}_A = \text{CPU 时钟周期数}_A\times \text{ 时钟周期长度} = I\times2.0\times250\text{ps} = 500\times I\text{ ps}$$

对于 B，同理有：

$$\text{CPU 时间}_B = I\times1.2\times500\text{ps} = 600\times I\text{ ps}$$

显然，计算机 A 更快。具体快多少可由执行时间之比来计算给出：

$$\frac{\text{CPU性能}_A}{\text{CPU性能}_B}=\frac{\text{执行时间}_B}{\text{执行时间}_A}=\frac{600\times I\text{ ps}}{500\times I\text{ ps}}=1.2$$

因此，对于该程序，计算机 A 的性能是计算机 B 的 1.2 倍。

1.6.5　经典的 CPU 性能公式

现在我们可以用**指令数**（程序执行所需要的指令总数）、CPI（指令平均时钟周期数）和时钟周期长度来写出基本的性能公式：

指令数：执行某程序所需的总指令数量。

$$\text{CPU 时间} = \text{指令数} \times \text{CPI} \times \text{时钟周期长度}$$

或考虑到时钟频率和时钟周期长度互为倒数，可写为：

$$\text{CPU时间} = \frac{\text{指令数} \times \text{CPI}}{\text{时钟频率}}$$

这些公式特别有用，因为它们将三个影响性能的关键因素进行了分离。如果知道实现方案或替代方案如何影响这三个参数，我们可用这些公式来比较不同的实现方案或评估某个设计的替代方案。

例题 | 代码片段的比较

编译器的设计人员试图为某计算机在两个代码序列之间选择更优的排列。硬件设计者给出了如下数据：

	每类指令的CPI		
	A	B	C
CPI	1	2	3

对于某特定高级语言语句的实现，两个代码序列所需的指令数量如下：

代码序列	每类指令的数量		
	A	B	C
1	2	1	2
2	4	1	1

请问哪个代码序列执行的指令数更多？哪个执行速度更快？每个代码序列的 CPI 分别是多少？

答案 | 代码序列 1 共执行 2+1+2=5 条指令。代码序列 2 共执行 4+1+1=6 条指令。所以，代码序列 1 执行的指令数更少。

基于指令数和 CPI，我们可以用 CPU 时钟周期数公式计算出每个代码序列的总时钟周期数：

$$\text{CPU时钟周期数} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

因此

$$\text{CPU 时钟周期数}_1 = (2\times1) + (1\times2) + (2\times3) = 2+2+6 = 10 \text{ 个周期}$$
$$\text{CPU 时钟周期数}_2 = (4\times1) + (1\times2) + (1\times3) = 4+2+3 = 9 \text{ 个周期}$$

故代码序列 2 更快，尽管它多执行了一条指令。由于代码序列 2 的总时钟周期数较少，而指令数较多，因此它一定具有较小的总 CPI。CPI 可使用如下公式计算：

$$\text{CPI} = \frac{\text{CPU时钟周期数}}{\text{指令数}}$$

$$CPI_1 = \frac{\text{CPU时钟周期数}_1}{\text{指令数}_1} = \frac{10}{5} = 2.0$$

$$CPI_2 = \frac{\text{CPU时钟周期数}_2}{\text{指令数}_2} = \frac{9}{6} = 1.5$$

重点 图 1-15 给出了计算机在不同层次上的性能指标及其测量单位。通过这些指标的组合可以计算出程序的执行时间（单位为秒）：

$$\text{执行时间} = \frac{\text{秒}}{\text{程序}} = \frac{\text{指令数}}{\text{程序}} \times \frac{\text{时钟周期数}}{\text{指令}} \times \frac{\text{秒}}{\text{时钟周期}}$$

性能的构成因素	测量单位
程序的CPU执行时间	程序执行的时间，以秒为单位
指令总数	程序执行的指令数目
指令平均时钟周期数（CPU）	每条指令平均执行的时钟周期数
时钟周期长度	每个时钟周期的长度，以秒为单位

图 1-15 基本的性能指标及其测量单位

需要铭记于心的是，时间是唯一对计算机性能进行测量的完整而可靠的指标。例如，对指令系统进行调整从而减少指令数目可能降低时钟周期长度或提高 CPI，从而抵消了指令数量改进所带来的效果。类似地，由于 CPI 与执行的指令类型相关，执行指令数最少的代码未必具有最快的执行速度。

如何确定性能公式中这些因素的值呢？我们可以通过运行程序来测量 CPU 的执行时间，并且计算机的说明书中通常介绍了时钟周期长度。难以测量的是指令数和 CPI。当然，如果确定了时钟频率和 CPU 执行时间，我们只需要知道指令数或者 CPI 两者之一，就可以依据性能公式计算出另一个。

可以使用体系结构仿真器等软件工具，预先执行程序来测量出指令数，也可以用大多数处理器中的硬件计数器来测量执行的指令数、平均 CPI 和性能损失源等。由于指令数量取决于计算机体系结构，并不依赖于计算机的具体实现，因而我们可以在不知道计算机全部实现细节的情况下对指令数进行测量。但是，CPI 与计算机的各种设计细节密切相关，包括存储系统和处理器结构（我们将在第 4、5 章中看到），以及应用程序中不同类型的指令所占的比例。因此，CPI 对于不同应用程序是不同的，对于相同指令系统的不同实现方式也是不同的。

指令分布：在一个或多个程序中，对指令的动态使用频度的评价指标。

上述例子表明，只用一种因素（如指令数）去评价性能是危险的。当比较两台计算机时必须考虑全部三个因素，它们组合起来才能确定执行时间。如果某个因素相同（如上例中的时钟频率），则必须考虑不同的因素才能确定性能的优劣。因为 CPI 根据**指令分布**（instruction mix）的不同而变化，所以即使时钟频率是相同的，也必须比较指令总数和 CPI。在本章最后的练习题中，有几道题目要求评价一系列计算机和编译器的改进对时钟频率、CPI 和指令数目的影响。在 1.10 节，我们将讨论一种常见的性能评价方式，由于并非全面考虑各种因素，这可能形成误导。

理解程序性能 程序的性能与算法、编程语言、编译器、体系结构以及实际的硬件有关。下表概括了这些组成部分是如何影响CPU性能公式中的各种因素的。

硬件或软件指标	影响什么	如何影响
算法	指令数，CPI	算法决定源程序执行指令的数目，从而也决定了CPU执行指令的数目。算法也可能通过使用较快或较慢的指令影响CPI。例如，当算法使用更多的除法运算时，将会导致CPI增大
编程语言	指令数，CPI	编程语言显然会影响指令数，因为编程语言中的语句必须翻译为指令，从而决定了指令数。编程语言也可影响CPI，例如，Java语言充分支持数据抽象，因此将进行间接调用，需要使用CPI较高的指令
编译器	指令数，CPI	因为编译器决定了源程序到计算机指令的翻译过程，所以编译器的效率既影响指令数又影响CPI。编译器的角色可能十分复杂，并以多种方式影响CPI
指令系统体系结构	指令数，时钟频率，CPI	指令系统体系结构影响CPU性能的所有三个方面，因为它影响完成某功能所需的指令数、每条指令的周期数以及处理器的时钟频率

详细阐述 也许你期望CPI最小值为1.0。在第4章我们将看到，有些处理器在每个时钟周期可对多条指令取指并执行。有些设计者用IPC（Instruction Per Clock Cycle）来代替指令平均执行周期数CPI。如一个处理器每时钟周期平均可执行2条指令，则它的IPC=2，CPI=0.5。

详细阐述 虽然时钟周期长度传统上是固定的，但是为了节省能量或暂时提升性能，当今的计算机可以使用不同的时钟频率，因此我们需要对程序使用平均时钟频率。例如，Intel Core i7处理器在过热之前可以暂时将时钟频率提高10%。Intel称之为快速模式（Turbo mode）。

自我检测 某Java程序在桌面处理器上运行需要15秒。一个新版本的Java编译器发行了，其编译产生的指令数量是旧版本Java编译器的0.6倍，不幸的是，CPI增加为原来的1.1倍。请问该程序在新版本的Java编译器中运行速度是多少？从以下三个选项中选出正确答案。

a. $\frac{1.5\times0.6}{1.1}$=8.2秒

b. 15×0.6×1.1=9.9秒

c. $\frac{1.5\times1.1}{0.6}$=27.5秒

1.7　功耗墙

图1-16描述了30年来Intel八代微处理器的时钟频率和功率的增长趋势。两者的快速增长几乎保持了几十年，但近几年来突然缓和下来。二者增长率保持同步的原因在于它们是密切相关的，而放缓的原因在于功率已经达到了实际极限，无法再将普通商用处理器冷却下来。

虽然功率决定了能够冷却的极限，然而在后PC时代，能量是真正关键的资源。对于个人移动设备来说，电池寿命比性能更为关键。对于具有100 000个服务器的仓储式计算机来说，冷却费用非常高，因此设计者要尽量降低其功率和冷却所带来的成本。在评价性能时，使用执行时间比使用MIPS（见1.10节）之类的比率更加可信，与此类似，在评价功耗时使

用焦耳这样的能量单位比瓦特这样的功率单位更加合理，可以为能耗采用焦耳 / 秒这样的评价单位。

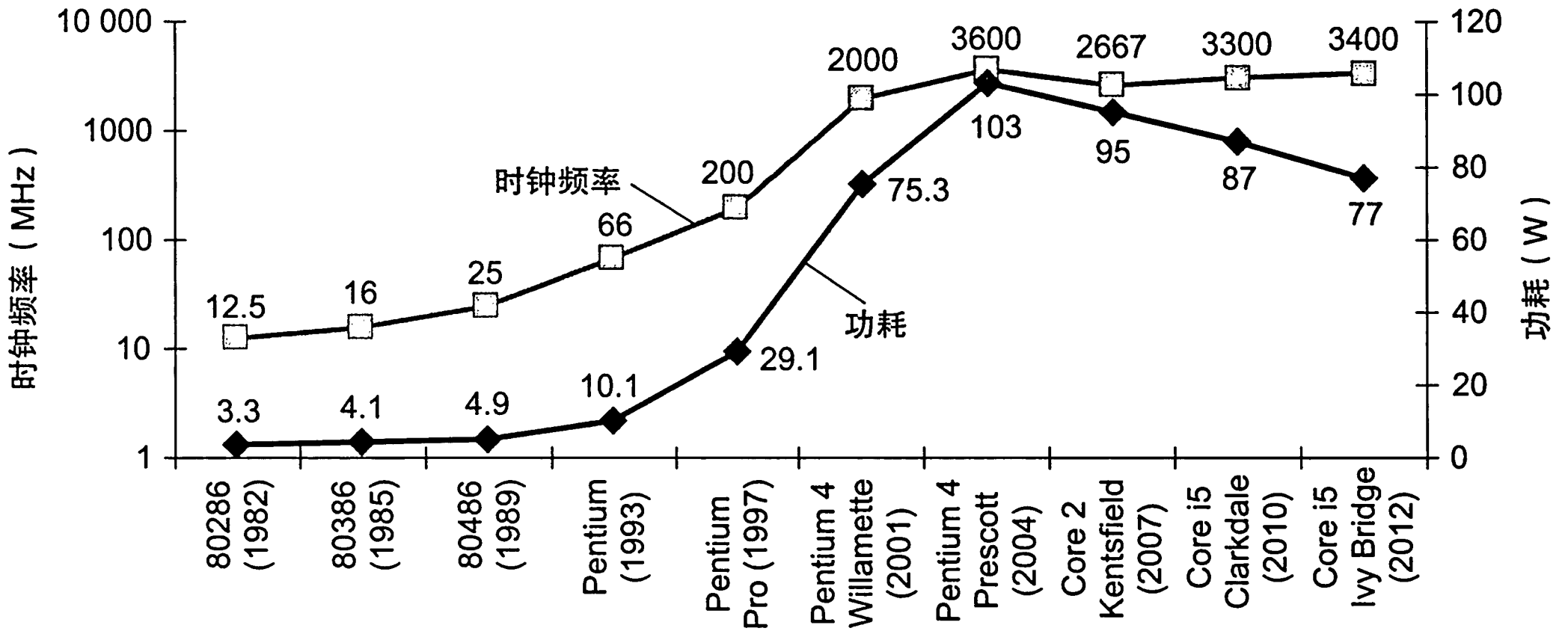

图 1-16　30 年间 Intel x86 八代微处理器的时钟频率和功耗。奔腾 4 处理器的时钟频率和功耗提高很大，但是性能提升不大。Prescott 发热问题导致奔腾 4 处理器的生产线被放弃。Core 2 生产线恢复使用低时钟频率的简单流水线和片上多处理器。Core i5 采用同样的流水线

当前在集成电路技术中占统治地位的是 CMOS（互补型金属氧化半导体），其主要的能耗来源是动态能耗，即在晶体管开关过程中产生的能耗，即晶体管的状态从 0 翻转到 1 或从 1 翻转到 0 消耗的能量。动态能耗取决于每个晶体管的负载电容和工作电压：

$$\text{能耗} \propto \text{负载电容} \times \text{电压}^2$$

这个等式表示的是一次 0 → 1 → 0 或 1 → 0 → 1 的逻辑转换过程中消耗的能量。一个晶体管消耗的能量为：

$$\text{能耗} \propto 1/2 \times \text{负载电容} \times \text{电压}^2$$

每个晶体管需要的功耗是一次翻转需要的能耗和开关频率的乘积：

$$\text{功耗} \propto 1/2 \times \text{负载电容} \times \text{电压}^2 \times \text{开关频率}$$

开关频率是时钟频率的函数，负载电容是连接到输出上的晶体管数量（称为扇出）和工艺的函数，该函数决定了导线和晶体管的电容。

思考一下图 1-16 的趋势，为什么时钟频率增长为 1000 倍，而功耗只增长了 30 倍呢？因为功率是电压平方的函数，功率和能耗能够通过降低电压来大幅减少，每次工艺更新换代时都会这样做。一般来说，每次技术更新换代可以使得电压降低大约 15%。20 多年来，电压从 5V 降到了 1V。这就是功耗只增长 30 倍的原因。

例题 | 相对功耗

假设我们需要开发一种新处理器，其负载电容只有复杂的旧处理器的 85%。再进一步假设其电压可以调节，与旧处理器相比电压降低了 15%，进而导致频率也降低了 15%，请问这对动态功耗有何影响？

答案 |

$$\frac{\text{功耗}_{\text{新}}}{\text{功耗}_{\text{旧}}} = \frac{(\text{电容负载} \times 0.85) \times (\text{电压} \times 0.85)^2 \times (\text{开关频率} \times 0.85)}{\text{电容负载} \times \text{电压}^2 \times \text{开关频率}}$$

于是功耗比为：

$$0.85^4=0.52$$

因此新处理器的功耗大约为旧处理器的一半。

目前的问题是如果电压继续下降会使晶体管的泄漏电流过大，就像水龙头不能被完全关闭一样。目前 40% 的功耗是由于泄漏电流造成的，如果晶体管的泄漏电流进一步增大，情况将会变得难以处理。

为了解决功耗问题，设计者已尝试连接大型设备以改善冷却效果，同时关闭芯片中在给定时钟周期内暂时不用的部分。尽管有很多更加昂贵的方式来冷却芯片，可以继续将芯片的功耗提升到如 300W 的水平，但这对于个人计算机甚至服务器来说成本太高了，个人移动设备就更不用说了。

由于计算机设计者遇到了功耗墙问题，因此他们需要开辟新的路径，选择不同于 30 年来设计微处理器的方式。

详细阐述 虽然动态能耗是 CMOS 能耗的主要来源，但静态能耗也是存在的，因为即使在晶体管关闭的情况下，也有泄漏电流存在。在服务器中，典型的电流泄漏占 40% 的能耗。因此，只要增加晶体管的数目，即使这些晶体管总是关闭的，也仍然会增加漏电能耗。人们采用各种各样的设计和工艺创新来控制电流泄漏，但还是难以进一步降低电压。

详细阐述 功耗成为集成电路设计的挑战有两个原因。首先，电源必须由外部输入并且分布到芯片的各个角落。现代微处理器通常使用几百个引脚作为电源和地线！同样，多层次芯片互联仅仅为了解决芯片的电源和地的分布比例问题。其次，功耗作为热量形式散发，因此必须进行散热处理。服务器芯片的功耗可高达 100W 以上，因此芯片及外围系统的散热是仓储规模计算机的主要开销（见第 6 章）。

1.8　沧海巨变：从单处理器向多处理器转变

功耗的极限迫使微处理器的设计产生了巨变。图 1-17 给出了桌面微处理器的程序响应时间的发展。从 2002 年起，其每年的增长速率从 1.5 下降到 1.2。

> 迄今为止，很多软件很像独奏者所写的音乐；使用当代的芯片，我们对于编写二重奏、四重奏以及小型合奏的经验不多，为大型交响乐或者合唱谱曲则是一种不同的挑战。
> *Brian Hayes, Computing in a Parallel Universe, 2007*

在 2006 年，所有桌面和服务器公司都在单个微处理器芯片中加入了多个处理器，以求更大的吞吐率，而不再继续追求降低单个程序运行在单个处理器上的响应时间。为了减少处理器（processor）和微处理器（microprocessor）这两个词语之间的混淆，一些公司将处理器作为“核”（core）的代称，这种语境下的微处理器就是多核处理器了。因此，“四核”微处理器是一个包含了 4 个处理器或者 4 个核的芯片。

在过去，程序员可以依赖于硬件、体系结构和编译器的创新，无须修改一行代码，就能实现程序的性能每 18 个月翻一番。而今天，程序员要想显著改进响应时间，必须重写程序以充分利用多处理器的优势。而且，随着核的数目不断加倍，程序员也必须不断改进代码，以便在新微处理器上获得显著的性能提升。

为了强调软件和硬件系统的协同工作，我们在本书中用“硬件 / 软件接口”的概念来进行描述，介绍一些重要的观点，下面是本书中的第一个。

硬件 / 软件接口 并行性对计算性能一直十分重要，但它往往是隐蔽的。第 4 章将说明流

水线，它是一种漂亮的技术，通过指令重叠执行使程序运行得更快。这是指令级并行的一个例子。在抽取了硬件的并行本质之后，程序员或编译器可认为在硬件中指令是串行执行的。

迫使程序员意识到硬件的并行性，并显式地按并行方式重写程序，曾经是计算机体系结构的“高压线”，以致很多采用此种方式进行革新的公司都失败了（见 6.15 节）。从这个历史的角度来观察，令人吃惊的是，整个信息技术行业已经认为未来程序员将成功切换到显式并行编程上。

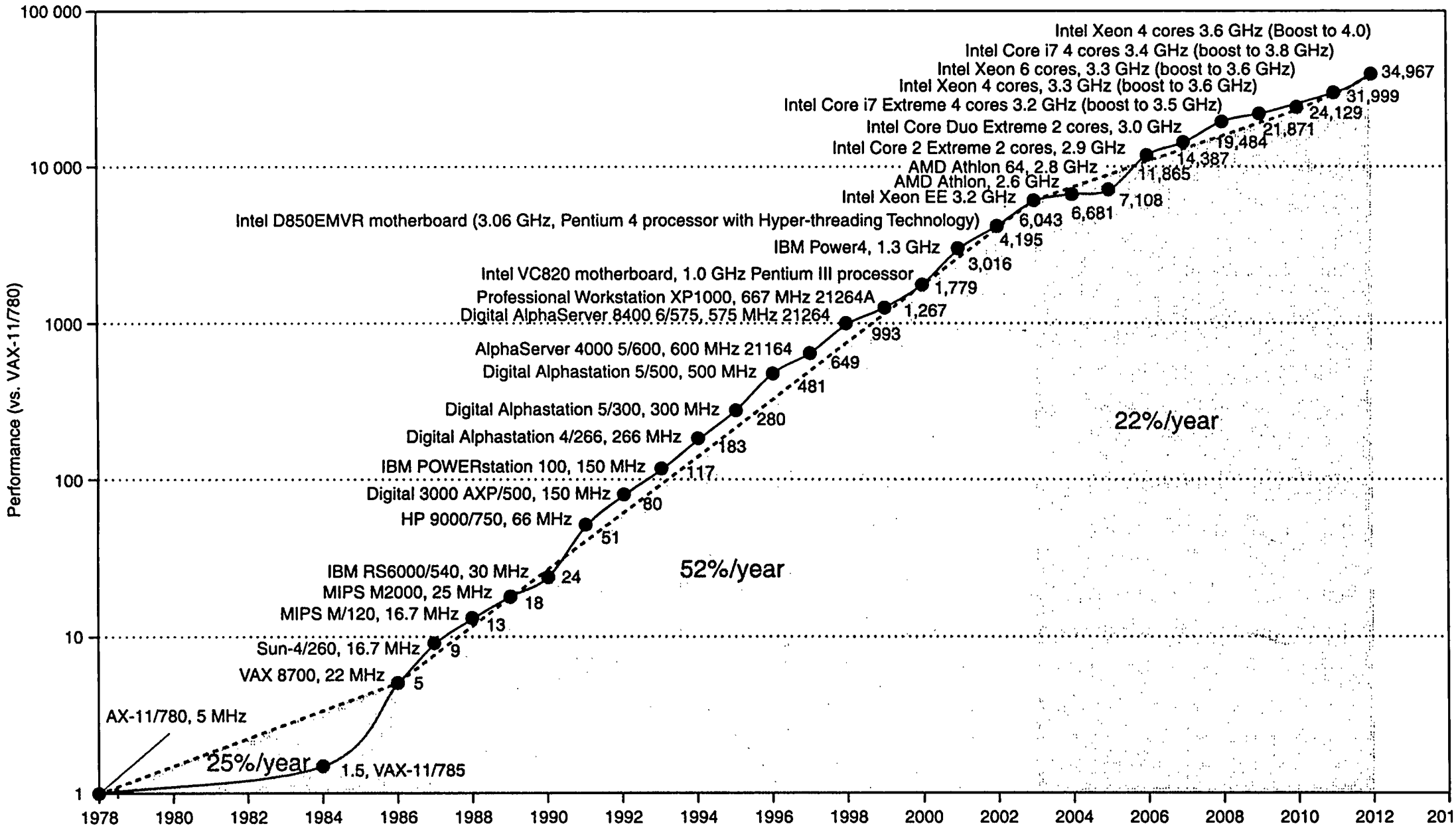

图 1-17 自 20 世纪 80 年代中期以来处理器性能的发展。本图描绘了和 VAX 11/780 相比，采用 SPECint 评测程序得到的性能数据（见 1.10 节）。在 20 世纪 80 年代中期以前，性能的增长主要靠技术驱动，平均每年增长 25%。在这个阶段之后，增长速度达到 52%，这归功于体系结构和组织方式的创新。从 20 世纪 80 年代中期开始，性能每年大约提高 52%，如果按照原先的 25% 的增长率计算，则到 2002 年的性能只有实际的 1/7。从 2002 年开始，受到功耗、指令级并行程度和存储器长延迟的限制，单核处理器的性能增长放缓，大约每年 22%

为什么程序员编写显式并行程序如此困难呢？第一个原因是并行编程以提高性能为目的，这必然增加编程的难度。不仅程序必须要正确，能够解决重要问题，而且运行速度要快，还需要为用户或其他程序提供接口以便使用。如果不关心性能的话，编写一个串行程序就足够了。

第二个原因是为了发挥并行硬件的速度，程序员必须将应用划分为每个核上有大致相同数量的任务，并同时完成。还要尽可能减少调度的开销，不浪费并行性能。

打个比方，现在有一个写新闻故事的任务，如果由八名记者共同来完成，能否提高八倍的写作速度呢？为了实现这一目标，这个新闻故事需要进行划分，让每个记者都有事可做。假如某名记者分到的任务比其他七名记者加起来的任务还要多，那用八名记者的好处就不存在了。因此，任务分配必须平衡才能得到预期的加速。另一个存在的危险是记者要花费时间互相交流才能完成所分配的任务。如果故事的一部分（例如结论）在所有其他部分完成之前

无法撰写，则缩短故事撰写时间的计划将会失败。所以，必须尽量减少通信和同步的开销。对于上述与并行编程的类比来说，其挑战包括调度、负载平衡、同步所需时间以及各部分间通信负载等问题。你也许会想到，当更多的记者来写一个故事时挑战会更大，类似的，当核的数目更多时并行编程的挑战也将更大。

为了反映业界的这个沧海巨变，后面的五章中每章都会至少有一节介绍有关并行的内容：

- 第 2 章，2.11 节。通常独立的并行任务需要一次次地协调，通报它们何时完成了所分配的任务。这一章将解释多核处理器任务同步时所使用的指令。
- 第 3 章，3.6 节。获取并行性的最简单方式是，将计算单元并行工作，例如两个向量相乘。摩尔定律提供了位宽更大且能同时处理多个操作数的算术单元，子字并行就利用了这种资源的并行性。
- 第 4 章，4.10 节。鉴于显式并行编程十分困难，在 20 世纪 90 年代人们付出了巨大的努力，从最初的流水线开始，让硬件和编译器可以发现隐含的并行性。这一章介绍了一些激进的技术，包括同时进行多指令的取指和执行，预测决策结果以及基于预测的推测式执行机制等。
- 第 5 章，5.10 节。降低通信开销的一个方法是让所有处理器使用统一的地址空间，由此使得任何处理器可以读写任何数据。当今的计算机都采用高速缓存技术，即在处理器附近更快的存储器中保存数据的一个临时副本。可以想象，如果多个处理器的高速缓存中的共享数据不一致，并行编程将尤为困难。这一章将介绍保持所有高速缓存数据一致性的机制。
- 第 5 章，5.11 节。这一节介绍如何使用多个磁盘共同构成一个能够提供更高吞吐率的系统，这就是廉价冗余磁盘阵列（RAID）的灵感。RAID 流行的真正原因是它能够通过采用适当数量的冗余磁盘提供更高的可靠性。这一节将介绍不同 RAID 级别的性能、成本和可靠性。

除了这些章节之外，还有一整章来介绍并行处理。第 6 章详细叙述了并行编程的挑战，提出了两种方法来解决：共享内存通信和显式消息传输，介绍了一种易于编程的并行性模型，讨论了使用基准评测程序对并行处理器进行评测的困难，为多核微处理器引入了一个新的简单性能模型，最后基于该模型对四种多核处理器进行了描述和评价。

如上所述，第 3 ～ 6 章使用矩阵向量相乘作为利用并行提高性能的例子。

附录 B 介绍了一种在桌面计算机中越来越普及的硬件部件：图形处理单元（Graphics Processing Unit，GPU）。它是为加速图像处理而发明的。得益于高度的并行性，GPU 表现出了优越的性能，并已发展为完善的编程平台。

附录 B 介绍了 NVIDIA GPU 及其并行编程环境的重要部分。

1.9　实例：评测 Intel Core i7

本书的每一章都有“实例”一节，它将本书中的概念与我们日常使用的计算机联系起来，这些小节涵盖了现代计算机中使用的技术。下面是本书中的第一个“实例”小节，我们将以 Intel Core i7 为例，说明如何制造集成电路，以及如何测量性能和功耗。

我想，就像书一样，“计算机”是一个全世界广泛应用的概念。但我没有想到它会发展得如此迅速，因为我完全没有预料到在一块芯片上可以得到像我们最终得到的如此多的部件。晶体管的进步完全出乎我们的预料，它比我们预想的发展要快。

J. Presper Eckert，ENIAC 的合作发明者，1991

1.9.1 SPEC CPU 基准评测程序

用户日复一日使用的程序是用于评价新型计算机最完美的程序。所运行的一组程序集构成了**工作负载**（workload）。要评价两台计算机系统，只需简单地比较工作负载在两台计算机上的执行时间。然而大多数用户并不这样做，他们通过其他方法测量计算机的性能，希望这些方法能够反映计算机执行用户工作负载的情况。最常用的测量方法是使用一组专门用于测量性能的**基准评测程序**（benchmark）。这些评测程序形成负载，用户期望预测实际负载的性能。我们在前面提到，要加速经常性事件的执行，必须先准确地知道哪些是经常性事件，因此基准评测程序在计算机体系结构中具有非常重要的作用。

工作负载：运行在计算机上的一组程序，可以直接采用用户的一组实际应用程序，也可以从实际程序中构建。一个典型的工作负载必须指明程序和相应的频率。

基准评测程序：遴选出来用于比较计算机性能的程序。

SPEC（System Performance Evaluation Cooperative）是由许多计算机销售商共同出资赞助并支持的合作组织，目的是为现代计算机系统建立基准评测程序集。1989 年，SPEC 建立了重点面向处理器性能的基准程序集（现在称为 SPEC89）。历经 5 代发展，目前最新的是 SPEC CPU2006，它包括 12 个整数基准程序集（CINT2006）和 17 个浮点基准程序集（CFP2006）。CINT2006 包括 C 编译器、量子计算机仿真、下象棋程序等，CFP2006 包括有限元模型结构化网格法、分子动力学质点法、流体动力学稀疏线性代数法等。

图 1-18 列举了 SPEC 整数基准程序及其在 Intel Core i7 上的执行时间、指令数、CPI 和时钟周期长度等组成的 SPEC 分值。注意，CPI 的最大值和最小值相差达到 5 倍。

描述	名称	指令数（×10⁹）	CPI	时钟周期长度（×10⁻⁹ 秒）	执行时间（秒）	参考时间（秒）	SPEC 分值
字符串处理程序	perl	2252	0.60	0.376	508	9770	19.2
块排序压缩	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C编译器	gcc	794	1.20	0.376	358	8050	22.5
组合优化	mcf	221	2.66	0.376	221	9120	41.2
围棋游戏（人工智能）	go	1274	1.10	0.376	527	10490	19.9
基因序列搜索	hmmer	2616	0.60	0.376	590	9330	15.8
国际象棋游戏（人工智能）	sjeng	1948	0.80	0.376	586	12100	20.7
量子计算机模拟	libquantum	659	0.44	0.376	109	20720	190.0
视频压缩	h264avc	3793	0.50	0.376	713	22130	31.0
离散事件模拟库	omnetpp	367	2.10	0.376	290	6250	21.5
游戏/寻径	astar	1250	1.00	0.376	470	7020	14.9
XML解析	xalancbmk	1045	0.70	0.376	275	6900	25.1
几何平均值	—	—	—	—	—	—	25.7

图 1-18 SPECINTC2006 基准程序在频率为 2.66GHz 的 Intel Core i7 920 上的运行结果。按照经典的 CPU 性能公式，执行时间是本表中三个因素的乘积：以亿为单位的指令数、每条指令的时钟数（CPI）以及纳秒级的时钟周期长度。SPEC 分值由 SPEC 提供，基于参考时间除以所测量的执行时间而得到。SPECINT2006 所引用的分值是所有 SPEC 分值的几何平均值

为了简化评测结果，SPEC 决定使用单一的数字来归纳所有 12 种整数基准程序。具体方

法是将被测计算机的执行时间标准化，即将参考处理器的执行时间除以被测计算机的执行时间，这样的归一化结果产生了一个测量值，称为 SPEC 分值。SPEC 分值越大，表示性能越好（因为 SPEC 分值和被测计算机的执行时间成反比）。CINT2006 或 CFP2006 的综合评测结果是取 SPEC 分值的几何平均值。

详细阐述 在使用 SPEC 分值比较两台计算机时采用的是几何平均值，这样可以使得无论采用哪台计算机进行标准化都可得到同样的相对值。如果采用的是算术平均值，结果会随选用的参照计算机而变。

几何平均值的公式是

$$\sqrt[n]{\prod_{i=1}^{n} \text{执行时间比}_i}$$

其中，执行时间比 $_i$ 是总计 n 个工作负载中第 i 个程序的执行时间按参考计算机进行标准化的结果，并且

$$\prod_{i=1}^{n} a_i \text{表示} a_1 \times a_2 \times \cdots \times a_n$$

1.9.2　SPEC 功耗基准评测程序

由于能耗和功耗日益重要，SPEC 增加了一组用于评估功耗的基准评测程序，它可以报告一段时间内服务器在不同负载水平下（以 10% 的比例递增）的功耗。与前面类似，图 1-19 给出了在基于 Intel Nehalem 处理器的服务器上的评测结果。

目标负载 (%)	性能 (ssj_ops)	平均功耗 (W)
100%	865 618	258
90%	786 688	242
80%	698 051	224
70%	607 826	204
60%	521 391	185
50%	436 757	170
40%	345 919	157
30%	262 071	146
20%	176 061	135
10%	86 784	121
0%	0	80
合计	4 787 166	1922
∑ssj_ops / ∑power		2490

图 1-19　SPECpower_ssj2008 在服务器上的运行结果。服务器的具体配置为双插槽 2.6GHz Intel Xeon X5650 处理器，配备 16GB DRAM 及 100GB 固态硬盘

SPECpower 最早来自面向 Java 商业应用的 SPEC 基准程序（SPECJBB2005），它主要评测处理器、高速缓存、主存以及 Java 虚拟机、编译器、垃圾回收器、操作系统等。性能采用吞吐率来测量，单位是每秒完成的任务数量。同样，为了简化结果，SPEC 采用单个的数字来对评测结果进行归纳，称为每瓦执行的服务器端 Java 操作数量（overallssj_opsper watt），该单一化评测指标的计算公式是：

$$\text{overall ssj\_ops per watt} = \frac{\sum_{i=0}^{10} \text{ssj\_ops}_i}{\sum_{i=0}^{10} \text{power}_i}$$

式中，ssj_ops_i 为工作负载在每 10% 增量处的性能，power_i 是对应的功耗。

1.10 谬误与陷阱

> 科学一定开始于神话和对神话的批判。
> *卡尔·波普尔爵士，The Philosophy of Science，1957*

本书中每一章都会有“谬误与陷阱”一节，其目的是说明我们在实际中经常遇到的误解，我们称之为谬误。当讨论谬误时，我们会举出一个反例。我们也讨论陷阱，即那些容易犯的错误。陷阱通常是指仅在有限的上下文中才正确的一般原理。本节旨在帮助你在设计或使用计算机时避免犯同样的错误。价格 / 性能谬误和陷阱曾使包括我们在内的许多计算机架构师掉入圈套。因此，本书在这方面绝不缺少相关案例。下面介绍本书的第一个陷阱，虽然它曾迷惑了许多设计者，却揭示了计算机设计中的一个重要关系。

陷阱：在改进计算机的某个方面时期望总性能的提高与改进大小成正比。

加速经常性事件的伟大思想所导致的令人泄气的结果困扰着软件和硬件设计人员。它提醒我们一个事件所需要的时间影响着改进它的机会。

用一个简单的设计问题就可以很好地对其进行说明。假设一个程序在一台计算机上运行需要 100 秒，其中 80 秒的时间用于乘法操作。如果要把该程序的运行速度提高 5 倍，乘法操作的速度应该改进多少？

> **Amdahl 定律**：阐述了“对于特定改进的性能提升可能由所使用的改进特征的数量所限制”的规则。它是“收益递减定律”的量化版本。

改进后的程序执行时间可用下面的 Amdahl 定律计算：

$$\text{改进后的执行时间} = \frac{\text{受改进影响的执行时间}}{\text{改进量}} + \text{不受影响的执行时间}$$

代入本例的数据进行计算：

$$\text{改进后的执行时间} = \frac{80}{n} + (100 - 80)$$

由于要求快 5 倍，新的执行时间应该是 20，则有：

$$20 = \frac{80}{n} + 20$$

$$0 = \frac{80}{n}$$

也就是说，如果乘法运算占总负载的 80%，则无论怎样改进乘法，也无法达到性能提高 5 倍的结果。针对特定情况的性能提升，受到被改进的特征所占比例的限制。这个概念在日常生活中被称为边际收益递减定律。

当我们知道一些功能所消耗的时间及其可能的加速比时，就可以使用 Amdahl 定律对性能提升进行预估。将 Amdahl 定律与 CPU 性能公式结合，是一种很方便的对性能改进进行评估的工具。在本章练习中有关于 Amdahl 定律的更详细讨论。

Amdahl 定律还被应用于讨论并行处理器数量的实际限制，我们将在第 6 章的“谬误与陷阱”一节中对其进行研究。

谬误：低利用率的计算机具有更低功耗。

由于服务器的工作负载是变化的，所以低利用率情况下的功率很重要。例如，谷歌的仓储式计算机中服务器利用率在大多数情况下位于 10% ～ 50% 之间，只有不到 1% 的时间达到 100% 的利用率。即使花费 5 年时间来研究如何很好地运行 SPECpower 基准评测程序，在 2012 年，根据最优结果进行配置的计算机中，10% 的工作负载情况下也会使用 33% 的峰值功耗。实际工作中的系统由于没有针对 SPECpower 进行配置，其结果肯定会更加糟糕。

由于服务器的工作负载差异大且消耗了很大比例峰值功耗，Luiz Barroso 和 Urs Holzle[2007] 提出需要对硬件重新进行设计以达到“按能量比例计算”。这就是说，如果未来的服务器中在 10% 的工作负载时仅使用 10% 的峰值功耗，将减少数据中心的电费，并在环保时代做出重要的节能减排贡献。

谬误：面向性能的设计和面向能效的设计具有不相关的目标。

由于能耗是功率和时间的乘积，在通常情况下，对于软硬件的性能优化而言，即使优化需要更多的能耗，但是这些优化缩短了系统运行时间，因此整体上也还是节约了能量。一个重要的原因在于，只要程序运行，计算机的其他部分就会消耗能量。因此，即使优化的部分多消耗了一些能量，运行时间缩短也可以降低整个系统的能耗。

陷阱：用性能公式的一个子集去度量性能。

我们之前就指出过只用时钟频率、指令数和 CPI 之一来预测性能的危险。而另一种常犯的错误是只用三种因素之二去比较性能。虽然这样做在有些限定场景下可能正确，但这种方法仍然容易被误用。实际上，几乎所有取代时间以度量性能的方法都会导致误导性的声明、扭曲的结果或错误的解释。

有一种取代时间以度量性能的尺度是**每秒百万条指令数**，即 MIPS。对于一个给定的程序，MIPS 可简单表示为：

MIPS：基于百万条指令的程序执行速度的一种度量。指令条数除以执行时间与 10^6 之积就得到了 MIPS。

$$\text{MIPS} = \frac{\text{指令数}}{\text{执行时间} \times 10^6}$$

MIPS 是指令执行的速率，它规定了性能与执行时间成反比，越快的计算机具有越高的 MIPS 值。MIPS 的优点是既容易理解，又符合人的直觉，机器越快则 MIPS 值越高。

但实际上使用 MIPS 作为度量性能的指标存在三个问题。首先，MIPS 规定了指令执行的速率，但没有考虑指令的能力。我们没有办法用 MIPS 比较不同指令系统的计算机，因为指令数肯定是不同的。其次，在同一计算机上，不同的程序会有不同的 MIPS，因而一台计算机不能拥有单一的 MIPS 分值。例如，将执行时间用 MIPS、CPI、时钟频率代入之后可得：

$$\text{MIPS} = \frac{\text{指令数}}{\frac{\text{指令数} \times \text{CPI}}{\text{时钟频率}} \times 10^6} = \frac{\text{时钟频率}}{\text{CPI} \times 10^6}$$

回顾一下，图 1-18 显示了 SPEC CPU 2006 在 Intel Core i7 上的 CPI 最大值和最小值是相差 5 倍的，这导致相应的 MIPS 也是如此。最后一点也是最重要的一点，如果一个新程序执行的指令数更多，但每条指令的执行速度更快，则 MIPS 可能独立于性能而发生变化！

自我检测 考虑某程序在两台计算机上的性能测量结果，如下表：

测量内容	计算机A	计算机B
指令数	100亿条	80亿条
时钟频率	4 GHz	4 GHz
CPI	1.0	1.1

a. 哪台计算机的 MIPS 值更高？

b. 哪台计算机更快？

1.11 本章小结

（哪里……）ENIAC 配备有 18 000 个真空管，重达 30 吨，未来的计算机可能只需 1000 个真空管，或许仅仅有 1.5 吨重。
大众机械，1949 年 3 月

虽然很难准确预测计算机的成本与性能在未来将发展到怎样的水平，但可以确定的是一定会比现在的计算机好得多。为了能够跟随这样的进步，计算机设计人员和程序员必须理解更广泛的问题。

硬件和软件设计者都采用分层的方法构建计算机系统，每个下层都对其上层隐藏本层的细节。抽象思想是理解当今计算机系统的基础，但这并不意味着设计者只要懂得抽象原理就足够了。也许最重要的抽象层次是硬件和底层软件之间的接口，称为*指令系统体系结构*。保持指令系统体系结构恒定，使得基于该指令系统体系结构的不同实现（可能在成本和性能上有所不同）能够运行相同的软件。这种方法的一个可能不足是，会阻止某些需要修改该接口的创新。

有一个可靠的测量并报告性能的方法，即用真实程序的执行时间作为尺度。该执行时间与我们能够通过下面公式测量到的其他重要指标相关：

$$\frac{\text{秒数}}{\text{程序}}=\frac{\text{指令数}}{\text{程序}}\times\frac{\text{时钟周期数}}{\text{指令数}}\times\frac{\text{秒数}}{\text{时钟周期数}}$$

在本书中我们将多次使用这一公式及其组成因子。但请记住，任何一个独立的因子都不能决定性能，只有三个因子的乘积，即执行时间才是可靠的性能度量标准。

重点 执行时间是唯一有效且不可推翻的性能度量指标。人们曾经提出许多其他度量指标，但均存在不足之处。有些从一开始就没有反映执行时间，因而是无效的；还有一些只能在有限条件下正确，超出了限制条件则失效，或是没有清晰地说明有效性的限制条件。

现代处理器的关键硬件技术是硅。与理解集成电路技术同样重要的是理解如摩尔定律所预测的技术进步速度。硅技术加快了硬件的进步，与此同时，计算机组织中的新思想也改进了计算机的性价比。这其中有两个重要的新思想：第一，开发程序中的并行性，当前的典型方法是借助多处理器；第二，开发存储器层次结构的访问局部性，当前的典型方法是使用高速缓存。

能效已经取代芯片面积成为微处理器设计中最重要的资源。在试图改进性能的同时节省功耗，这样的需求已经迫使硬件行业转向多核微处理器，从而要求软件行业转向并行编程。并行化现在是提高性能的必要途径。

计算机设计总是以价格和性能来度量的，也包括其他一些重要的因素，如能耗、可靠性、成本及可扩展性等。尽管本章的重点在于价格、性能和能耗，但是最佳的设计应该在特定的应用领域中取得所有因素之间适当的平衡。