C++程序只需要表现为“好像”语句按顺序执行。C++编译器和计算机本身可能会改变执行顺序，以使程序运行得更快，只要任何计算的意义没有改变。

从C++11开始，C++不再认为只有一个执行地址。C++标准库现在提供了启动和停止线程以及在线程之间同步访问内存的工具。在C++11之前，程序员向C++编译器谎报线程，有时会导致难以调试的问题。

·某些存储器地址可以是设备寄存器而不是普通存储器。这些地址的值可能会在同一线程两次连续读取位置之间的时间间隔内发生变化，这表明硬件发生了一些变化。这样的位置在C++中被描述为volatile。声明变量volatile需要编译器每次获取变量的新副本使用，而不是通过将值保存在寄存器中并重用它来优化程序。指向易失性存储器的指针也可以声明。

C++11提供了一个叫做std：：atomic<>的魔法咒语，它可以让内存在一段时间内表现得就像一个简单的线性字节存储器一样，希望摆脱现代微处理器的所有复杂性，包括多线程执行，多层内存缓存等等。一些开发人员认为这就是volatile的作用。他们错了。

优化的含义是，对内存的访问支配着处理器中的其他成本，包括执行指令的成本。

图形用户界面, 应用程序, Teams

描述已自动生成

当执行单元需要获取不在该高速缓存中的数据时，必须丢弃当前在高速缓存中的一些数据以腾出空间。被选择丢弃的数据通常是最近最少使用的数据。这对优化很重要，因为这意味着频繁使用的内存位置可以比不太频繁使用的位置更快地访问。

，这意味着包含循环的代码块可能会执行得更快，因为组成循环的指令被大量使用并且相互靠近，因此可能会留在缓存中。包含函数调用的代码块或导致执行跳来跳去的状态可能会执行得更慢，因为代码的各个部分使用得不太频繁，并且不靠近。

* 大端（Big-Endian）：在这种模式下，数据的最高有效位（MSB，即最左边的一位）存储在内存的最低地址处。例如，整数0x01234567在内存中的存储顺序是0x01, 0x23, 0x45, 0x67。
* 小端（Little-Endian）：与小端模式相反，数据的最低有效位（LSB，即最右边的一位）存储在内存的最低地址处。同样地，整数0x01234567在小端模式下的存储顺序是0x67, 0x45, 0x23, 0x01。

如果操作系统在同一个程序中从一个线程切换到另一个线程，这意味着为被挂起的线程保存处理器的寄存器，并为被恢复的线程加载保存的寄存器。现代处理器的寄存器包含数百字节的数据。当新线程恢复执行时，它的数据可能不在缓存中，因此在新上下文加载到缓存中时，会有一段缓慢执行的初始时间。因此，切换线程上下文的成本相当高。

优化意义：由于决策（如分支和跳转）可能导致流水线停顿，因此优化程序以减少这些决策的频率可以提高性能。例如，通过减少不必要的分支指令或使用分支预测技术，可以减少流水线的停顿时间，从而提高程序的执行速度

当操作系统将上下文从一个程序切换到另一个程序时，过程甚至更加昂贵。所有脏缓存页（写入数据尚未到达主内存的页）必须刷新到物理内存。所有处理器寄存器都被保存。

* 优化的重要性：由于C++的这些特性，优化C++代码时，不能仅仅根据语句的表面形式来判断其性能开销。开发者需要了解背后的对象结构和运算符重载的实现细节，以便进行有效的优化。例如，避免不必要的对象复制和优化重载运算符的实现可以显著提高性能。
* **内存访问是处理器中其他开销的主导因素**。
* **未对齐的内存访问耗时是对齐访问的两倍**。
* **频繁访问的内存位置比不常访问的位置访问速度更快**。
* **相邻的内存位置比分散的内存位置访问速度更快**。
* **由于缓存的存在，一个函数在整个程序上下文中运行可能比在测试环境中运行更慢**。
* **访问线程间共享的数据比访问非共享数据慢得多**。
* **计算比决策更快**。
* **每个程序都在与其他程序竞争计算机资源**。
* **如果程序必须在启动时或高峰负载时运行，则必须在负载下测量性能**。
* **每次赋值、函数参数初始化和函数返回都会调用构造函数，而构造函数可能隐藏任意数量的代码**。
* **某些语句隐藏了大量的计算。语句的形式并不能反映其实际的开销**。
* **同步代码会减少并发线程共享数据时可获得的并发量**。

Amdahl定律

其中 ST 是由于优化导致的整个程序运行时间的改进比例，P 是受到优化的总原始运行时间的比例，Sp​ 是经过优化的部分 *P* 的改进比例。

解释：

这个公式用来计算优化对整个程序运行时间的影响。具体来说，它考虑了优化只影响到程序的一部分运行时间的情况，并且这部分运行时间的优化程度可能与整体的优化程度不同。

* ST*ST*​ 表示的是整个程序运行时间的改进比例。它是我们想要计算的量。
* (1−*P*) 表示没有受到优化的那部分运行时间所占的比例。
* P/Sp表示受到优化的那部分运行时间所占的比例，考虑到优化的效果后，这一部分的运行时间减少了 *Sp*​ 倍。
* 公式的分母是将这两部分相加，表示总的运行时间的变化情况。

优秀的优化者（像所有优秀的科学家一样）关注可重复性。这就是实验室笔记本的用武之地。每次计时运行都以一个假设、一个或多个小的代码调整和一个输入数据集开始，并以一个不起眼的整数毫秒结束。记住上一次运行的时间并不难，以便与下一次进行比较。只要每一次代码更改都成功，就足够了。

衡量基准绩效并设定目标

优化工作由两个数字指导：优化前的性能基线测量和性能目标。

启动时间

从按下回车键到程序进入主输入处理循环所用的时间。通常，但并不总是，开发人员可以只测量进入main（）过程和进入主循环之间所用的时间。提供程序认证的操作系统供应商对计算机启动时或用户登录时运行的程序有严格的要求。例如，微软要求寻求认证的硬件供应商Windows shell必须在启动后10秒内进入主循环。这限制了供应商在忙碌的启动环境中允许预加载和启动的其他程序的数量。Microsoft提供专门的工具来测量启动时间。

因为重新启动服务或长时间运行的程序所需的时间等于其关闭时间加上其启动时间。关闭时间从用户单击“关闭”图标或输入退出命令到进程实际退出所用的时间。通常（但不总是），您可以测量从主窗口接收到关闭命令到退出main（）所用的时间。关闭时间还包括停止所有线程和相关进程所需的时间。提供认证的操作系统供应商对关机时间有严格要求。关闭时间也很重要，因为重新启动服务或长时间运行的程序所需的时间等于其关闭时间加上其启动时间。

响应时间

执行命令所需的平均或最坏情况时间。对于网站来说，平均响应时间和最坏情况响应时间都有助于用户对网站的满意度。响应时间可以分为以下粗略的-10次幂篮子：

小于0.1秒：用户直接控制

0.1到1秒：用户控制命令

1到10秒：计算机处于控制状态

10秒以上：休息时间

你只能改善你所衡量的

配置文件程序执行

分析仪仪器经过精心设计，尽可能便宜。它对整体运行时间的影响很小，通常会使每个操作的速度降低百分之几。第一种方法给出精确的数字，代价是更高的开销和禁用某些优化。第二种方法给出了近似的结果，可能会错过一些不经常调用的函数，但具有在生产代码上运行的优势。

文本, 信件

描述已自动生成