```
    t += counter++;     /* +1 */
    t += counter++;     /* +2 */
    t += counter++;     /* +3 */
    return t;
}
```

这样的转换既减少了函数调用的开销，也允许对展开的代码做进一步优化。例如，编译器可以统一 func1in 中对全局变量 counter 的更新，产生这个函数的一个优化版本：

```
/* Optimization of inlined code */
long func1opt() {
    long t = 4 * counter + 6;
    counter += 4;
    return t;
}
```

对于这个特定的函数 f 的定义，上述代码忠实地重现了 func1 的行为。

GCC 的最近版本会尝试进行这种形式的优化，要么是被用命令行选项“-finline”指示时，要么是使用优化等级-O1 或者更高的等级时。遗憾的是，GCC 只尝试在单个文件中定义的函数的内联。这就意味着它将无法应用于常见的情况，即一组库函数在一个文件中被定义，却被其他文件内的函数所调用。

在某些情况下，最好能阻止编译器执行内联替换。一种情况是用符号调试器来评估代码，比如 GDB，如 3.10.2 节描述的一样。如果一个函数调用已经用内联替换优化过了，那么任何对这个调用进行追踪或设置断点的尝试都会失败。还有一种情况是用代码剖析的方式来评估程序性能，如 5.14.1 节讨论的一样。用内联替换消除的函数调用是无法被正确剖析的。

在各种编译器中，就优化能力来说，GCC 被认为是胜任的，但是并不是特别突出。它完成基本的优化，但是它不会对程序进行更加“有进取心的”编译器所做的那种激进变换。因此，使用 GCC 的程序员必须花费更多的精力，以一种简化编译器生成高效代码的任务的方式来编写程序。

5.2　表示程序性能

我们引入度量标准*每元素的周期数*(Cycles Per Element，CPE)，作为一种表示程序性能并指导我们改进代码的方法。CPE 这种度量标准帮助我们在更细节的级别上理解迭代程序的循环性能。这样的度量标准对执行重复计算的程序来说是很适当的，例如处理图像中的像素，或是计算矩阵乘积中的元素。

处理器活动的顺序是由时钟控制的，时钟提供了某个频率的规律信号，通常用*千兆赫兹*(GHz)，即十亿周期每秒来表示。例如，当表明一个系统有“4GHz”处理器，这表示处理器时钟运行频率为每秒 4×10^9 个周期。每个时钟周期的时间是时钟频率的倒数。通常是以*纳秒*(nanosecond，1 纳秒等于 10^{-9} 秒)或*皮秒*(picosecond，1 皮秒等于 10^{-12} 秒)为单位的。例如，一个 4GHz 的时钟其周期为 0.25 纳秒，或者 250 皮秒。从程序员的角度来看，用时钟周期来表示度量标准要比用纳秒或皮秒来表示有帮助得多。用时钟周期来表示，度量值表示的是执行了多少条指令，而不是时钟运行得有多快。

许多过程含有在一组元素上迭代的循环。例如，图 5-1 中的函数 psum1 和 psum2 计算的都是一个长度为 n 的向量的*前置和*(prefix sum)。对于向量 $\vec{a}=\langle a_0, a_1, \cdots, a_{n-1}\rangle$，前置和 $\vec{p}=\langle p_0, p_1, \cdots, p_{n-1}\rangle$ 定义为