回高速缓存引起的传送比较少，它允许更多的到内存的带宽用于执行 DMA 的 I/O 设备。此外，越往层次结构下面走，传送时间增加，减少传送的数量就变得更加重要。一般而言，高速缓存越往下层，越可能使用写回而不是直写。

旁注　高速缓存行、组和块有什么区别？

很容易混淆高速缓存行、组和块之间的区别。让我们来回顾一下这些概念，确保概念清晰：

- 块是一个固定大小的信息包，在高速缓存和主存(或下一层高速缓存)之间来回传送。
- 行是高速缓存中的一个容器，存储块以及其他信息(例如有效位和标记位)。
- 组是一个或多个行的集合。直接映射高速缓存中的组只由一行组成。组相联和全相联高速缓存中的组是由多个行组成的。

在直接映射高速缓存中，组和行实际上是等价的。不过，在相联高速缓存中，组和行是很不一样的，这两个词不能互换使用。

因为一行总是存储一个块，术语“行”和“块”通常互换使用。例如，系统专家总是说高速缓存的“行大小”，实际上他们指的是块大小。这样的用法十分普遍，只要你理解块和行之间的区别，它不会造成任何误会。

6.5　编写高速缓存友好的代码

在 6.2 节中，我们介绍了局部性的思想，而且定性地谈了一下什么会具有良好的局部性。明白了高速缓存存储器是如何工作的，我们就能更加准确一些了。局部性比较好的程序更容易有较低的不命中率，而不命中率较低的程序往往比不命中率较高的程序运行得更快。因此，从具有良好局部性的意义上来说，好的程序员总是应该试着去编写高速缓存友好(cache friendly)的代码。下面就是我们用来确保代码高速缓存友好的基本方法。

1) 让最常见的情况运行得快。程序通常把大部分时间都花在少量的核心函数上，而这些函数通常把大部分时间都花在了少量循环上。所以要把注意力集中在核心函数里的循环上，而忽略其他部分。

2) 尽量减小每个循环内部的缓存不命中数量。在其他条件(例如加载和存储的总次数)相同的情况下，不命中率较低的循环运行得更快。

为了看看实际上这是怎么工作的，考虑 6.2 节中的函数 sumvec：

```
1   int sumvec(int v[N])
2   {
3       int i, sum = 0;
4
5       for (i = 0; i < N; i++)
6           sum += v[i];
7       return sum;
8   }
```

这个函数高速缓存友好吗？首先，注意对于局部变量 i 和 sum，循环体有良好的时间局部性。实际上，因为它们都是局部变量，任何合理的优化编译器都会把它们缓存在寄存器文件中，也就是存储器层次结构的最高层中。现在考虑一下对向量 v 的步长为 1 的引用。一般而言，如果一个高速缓存的块大小为 B 字节，那么一个步长为 k 的引用模式(这里 k 是以字为单位的)平均每次循环迭代会有 $\min(1, (\text{wordsize} \times k)/B)$ 次缓存不命中。当 $k=1$ 时，它取最小值，所以对 v 的步长为 1 的引用确实是高速缓存友好的。例如，假设 v 是块对齐的，字为 4 个字节，高速缓存块为 4 个字，而高速缓存初始为空(冷高速缓存)。然