基本级别之外调整性能需要一些对处理器微体系结构的理解，描述处理器用来实现它的指令集体系结构的底层机制。对于乱序处理器的情况，只需要知道一些关于操作、容量、延迟和功能单元发射时间的信息，就能够基本地预测程序的性能了。

我们研究了一系列技术，包括循环展开、创建多个累积变量和重新结合，它们可以利用现代处理器提供的指令级并行。随着对优化的深入，研究产生的汇编代码以及试着理解机器如何执行计算变得重要起来。确认由程序中的数据相关决定的关键路径，尤其是循环的不同迭代之间的数据相关，会收获良多。我们还可以根据必须要计算的操作数量以及执行这些操作的功能单元的数量和发射时间，计算一个计算的吞吐量界限。

包含条件分支或与内存系统复杂交互的程序，比我们最开始考虑的简单循环程序，更难以分析和优化。基本策略是使分支更容易预测，或者使它们很容易用条件数据传送来实现。我们还必须注意存储和加载操作。将数值保存在局部变量中，使得它们可以存放在寄存器中，这会很有帮助。

当处理大型程序时，将注意力集中在最耗时的部分变得很重要。代码剖析程序和相关的工具能帮助我们系统地评价和改进程序性能。我们描述了 GPROF，一个标准的 Unix 剖析工具。还有更加复杂完善的剖析程序可用，例如 Intel 的 VTUNE 程序开发系统，还有 Linux 系统基本上都有的 VALGRIND。这些工具可以在过程级分解执行时间，估计程序每个基本块(basic block)的性能。(基本块是内部没有控制转移的指令序列，因此基本块总是整个被执行的。)

参考文献说明

我们的关注点是从程序员的角度描述代码优化，展示如何使书写的代码能够使编译器更容易地产生高效的代码。Chellappa、Franchetti 和 Püschel 的扩展的论文[19]采用了类似的方法，但关于处理器的特性描述得更详细。

有许多著作从编译器的角度描述了代码优化，形式化描述了编辑器可以产生更有效代码的方法。Muchnick 的著作被认为是最全面的[80]。Wadleigh 和 Crawford 的关于软件优化的著作[115]覆盖了一些我们已经谈到的内容，不过它还描述了在并行机器上获得高性能的过程。Mahlke 等人的一篇比较早期的论文[75]，描述了几种为编译器开发的将程序映射到并行机器上的技术，它们是如何能够被改造成利用现代处理器的指令级并行的。这篇论文覆盖了我们讲过的代码变换，包括循环展开、多个累积变量(他们称之为累积变量扩展(accumulator variable expansion))和重新结合(他们称之为树高度减少(tree height reduction))。

我们对乱序处理器的操作的描述相当简单和抽象。可以在高级计算机体系结构教科书中找到对通用原则更完整的描述，例如 Hennessy 和 Patterson 的著作[46，第 2～3 章]。Shen 和 Lipasti 的书[100]提供了对现代处理器设计深入的论述。

家庭作业

**5.13　假设我们想编写一个计算两个向量 u 和 v 内积的过程。这个函数的一个抽象版本对整数和浮点数类型，在 x86-64 上 CPE 等于 14～18。通过进行与我们将抽象程序 combine1 变换为更有效的 combine4 相同类型的变换，我们得到如下代码：

```
1   /* Inner product.  Accumulate in temporary */
2   void inner4(vec_ptr u, vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(u);
6       data_t *udata = get_vec_start(u);
7       data_t *vdata = get_vec_start(v);
8       data_t sum = (data_t) 0;
9
10      for (i = 0; i < length; i++) {
11          sum = sum + udata[i] * vdata[i];
12      }
13      *dest = sum;
14  }
```