5.11　一些限制因素

我们已经看到在一个程序的数据流图表示中，关键路径指明了执行该程序所需时间的一个基本的下界。也就是说，如果程序中有某条数据相关链，这条链上的所有延迟之和等于 T，那么这个程序至少需要 T 个周期才能执行完。

我们还看到功能单元的吞吐量界限也是程序执行时间的一个下界。也就是说，假设一个程序一共需要 N 个某种运算的计算，而微处理器只有 C 个能执行这个操作的功能单元，并且这些单元的发射时间为 I。那么，这个程序的执行至少需要 $N \cdot I/C$ 个周期。

在本节中，我们会考虑其他一些制约程序在实际机器上性能的因素。

5.11.1　寄存器溢出

循环并行性的好处受汇编代码描述计算的能力限制。如果我们的并行度 p 超过了可用的寄存器数量，那么编译器会诉诸溢出(spilling)，将某些临时值存放到内存中，通常是在运行时堆栈上分配空间。举个例子，将 combine6 的多累积变量模式扩展到 $k=10$ 和 $k=20$，其结果的比较如下表所示：

函数	方法	整数		浮点数	
		+	*	+	*
combine6					
	10×10 循环展开	0.55	1.00	1.01	0.52
	20×20 循环展开	0.83	1.03	1.02	0.68
吞吐量界限		0.50	1.00	1.00	0.50

我们可以看到对这种循环展开程度的增加没有改善 CPE，有些甚至还变差了。现代 x86-64 处理器有 16 个寄存器，并可以使用 16 个 YMM 寄存器来保存浮点数。一旦循环变量的数量超过了可用寄存器的数量，程序就必须在栈上分配一些变量。

例如，下面的代码片段展示了在 10×10 循环展开的内循环中，累积变量 acc0 是如何更新的：

```
Updating of accumulator acc0 in 10 x 10 urolling
vmulsd  (%rdx), %xmm0, %xmm0      acc0 *= data[i]
```

我们看到该累积变量被保存在寄存器%xmm0 中，因此程序可以简单地从内存中读取 data[i]，并与这个寄存器相乘。

与之相比，20×20 循环展开的相应部分非常不同：

```
Updating of accumulator acc0 in 20 x 20 unrolling
vmovsd  40(%rsp), %xmm0
vmulsd  (%rdx), %xmm0, %xmm0
vmovsd  %xmm0, 40(%rsp)
```

累积变量保存为栈上的一个局部变量，其位置距离栈指针偏移量为 40。程序必须从内存中读取两个数值：累积变量的值和 data[i]的值，将两者相乘后，将结果保存回内存。

一旦编译器必须要诉诸寄存器溢出，那么维护多个累积变量的优势就很可能消失。幸运的是，x86-64 有足够多的寄存器，大多数循环在出现寄存器溢出之前就将达到吞吐量限制。