够每个时钟周期开始一个新的操作，但是它只会每 L 个周期开始一条新操作，这里 L 是合并操作的延迟。现在我们要考察打破这种顺序相关，得到比延迟界限更好性能的方法。

5.9.1 多个累积变量

对于一个可结合和可交换的合并运算来说，比如说整数加法或乘法，我们可以通过将一组合并运算分割成两个或更多的部分，并在最后合并结果来提高性能。例如，P_n 表示元素 a_0，a_1，…，a_{n-1} 的乘积：

$$P_n = \prod_{i=0}^{n-1} a_i$$

假设 n 为偶数，我们还可以把它写成 $P_n = PE_n \times PO_n$，这里 PE_n 是索引值为偶数的元素的乘积，而 PO_n 是索引值为奇数的元素的乘积：

$$PE_n = \prod_{i=0}^{n/2-1} a_{2i}$$

$$PO_n = \prod_{i=0}^{n/2-1} a_{2i+1}$$

图 5-21 展示的是使用这种方法的代码。它既使用了两次循环展开，以使每次迭代合并更多的元素，也使用了两路并行，将索引值为偶数的元素累积在变量 acc0 中，而索引值为奇数的元素累积在变量 acc1 中。因此，我们将其称为“2×2 循环展开”。同前面一样，我们还包括了第二个循环，对于向量长度不为 2 的倍数时，这个循环要累积所有剩下的数组元素。然后，我们对 acc0 和 acc1 应用合并运算，计算最终的结果。

```
/* 2 x 2 loop unrolling */
void combine6(vec_ptr v, data_t *dest)
{
    long i;
    long length = vec_length(v);
    long limit = length-1;
    data_t *data = get_vec_start(v);
    data_t acc0 = IDENT;
    data_t acc1 = IDENT;

    /* Combine 2 elements at a time */
    for (i = 0; i < limit; i+=2) {
        acc0 = acc0 OP data[i];
        acc1 = acc1 OP data[i+1];
    }

    /* Finish any remaining elements */
    for (; i < length; i++) {
        acc0 = acc0 OP data[i];
    }
    *dest = acc0 OP acc1;
}
```

图 5-21 运用 2×2 循环展开。通过维护多个累积变量，这种方法利用了多个功能单元以及它们的流水线能力

比较只做循环展开和既做循环展开同时也使用两路并行这两种方法，我们得到下面的性能：

函数	方法	整数		浮点数	
		+	*	+	*
combine4	在临时变量中累积	1.27	3.01	3.01	5.01
combine5	2×1 展开	1.01	3.01	3.01	5.01
combine6	2×2 展开	0.81	1.51	1.51	2.51
延迟界限		1.00	3.00	3.00	5.00
吞吐量界限		0.50	1.00	1.00	0.50

我们看到所有情况都得到了改进，整数乘、浮点加、浮点乘改进了约 2 倍，而整数加也有所改进。最棒的是，我们打破了由延迟界限设下的限制。处理器不再需要延迟一个加法或乘法操作以待前一个操作完成。

要理解 combine6 的性能，我们从图 5-22 所示的代码和操作序列开始。通过图 5-23