


总的来说,重新结合变换能够减少计算中关键路径上操作的数量,通过更好地利用功能单元的流水线能力得到更好的性能。大多数编译器不会尝试对浮点运算做重新结合,因为这些运算不保证是可结合的。当前的 GCC 版本会对整数运算执行重新结合,但不是总有好的效果。通常,我们发现循环展开和并行地累积在多个值中,是提高程序性能的更可靠的方法。

 **练习题 5.8** 考虑下面的计算  $n$  个双精度数组组成的数组乘积的函数。我们 3 次展开这个循环。

```
double aprod(double a[], long n)
{
    long i;
    double x, y, z;
    double r = 1;
    for (i = 0; i < n-2; i+= 3) {
        x = a[i]; y = a[i+1]; z = a[i+2];
        r = r * x * y * z; /* Product computation */
    }
    for (; i < n; i++)
        r *= a[i];
    return r;
}
```

对于标记为 Product computation 的行,可以用括号得到该计算的五种不同的结合,如下所示:

```
r = ((r * x) * y) * z; /* A1 */
r = (r * (x * y)) * z; /* A2 */
r = r * ((x * y) * z); /* A3 */
r = r * (x * (y * z)); /* A4 */
r = (r * x) * (y * z); /* A5 */
```

假设在一台浮点数乘法延迟为 5 个时钟周期的机器上运行这些函数。确定由乘法的数据相关限定的 CPE 的下界。(提示:画出每次迭代如何计算  $r$  的图形化表示会所帮助。)

### 网络旁注 OPT:SIMD 用向量指令达到更高的并行度

就像在 3.1 节中讲述的,Intel 在 1999 年引入了 SSE 指令, SSE 是“Streaming SIMD Extensions(流 SIMD 扩展)”的缩写,而 SIMD(读作“sim-dee”)是“Single-Instruction, Multiple-Data(单指令多数据)”的缩写。SSE 功能历经几代,最新的版本为高级向量扩展(advanced vector extension)或 AVX。SIMD 执行模型是用单条指令对整个向量数据进行操作。这些向量保存在一组特殊的向量寄存器(vector register)中,名字为  $\%ymm0 \sim \%ymm15$ 。目前的 AVX 向量寄存器长为 32 字节,因此每一个都可以存放 8 个 32 位数或 4 个 64 位数,这些数据既可以是整数也可以是浮点数。AVX 指令可以对这些寄存器执行向量操作,比如并行执行 8 组数值或 4 组数值的加法或乘法。例如,如果 YMM 寄存器  $\%ymm0$  包含 8 个单精度浮点数,用  $a_0, \dots, a_7$  表示,而  $\%rcx$  包含 8 个单精度浮点数的内存地址,用  $b_0, \dots, b_7$  表示,那么指令

```
vmulps (%rcx), %ymm0, %ymm1
```

会从内存中读出 8 个值,并行地执行 8 个乘法,计算  $a_i \leftarrow a_i \cdot b_i, 0 \leq i \leq 7$ ,并将得到的