所示的过程，可以推导出一个模板，给出迭代之间的数据相关。同 combine5 一样，这个内循环包括两个 vmulsd 运算，但是这些指令被翻译成读写不同寄存器的 mul 操作，它们之间没有数据相关(图 5-23b)。然后，把这个模板复制 $n/2$ 次(图 5-24)，就是在一个长度为 n 的向量上执行这个函数的模型。可以看到，现在有两条关键路径，一条对应于计算索引为偶数的元素的乘积(程序值 acc0)，另一条对应于计算索引为奇数的元素的乘积(程序值 acc1)。每条关键路径只包含 $n/2$ 个操作，因此导致 CPE 大约为 5.00/2＝2.50。相似的分析可以解释我们观察到的对于不同的数据类型和合并运算的组合，延迟为 L 的操作的 CPE 等于 $L/2$。实际上，程序正在利用功能单元的流水线能力，将利用率提高到 2 倍。唯一的例外是整数加。我们已将将 CPE 降低到 1.0 以下，但是还是有太多的循环开销，而无法达到理论界限 0.50。

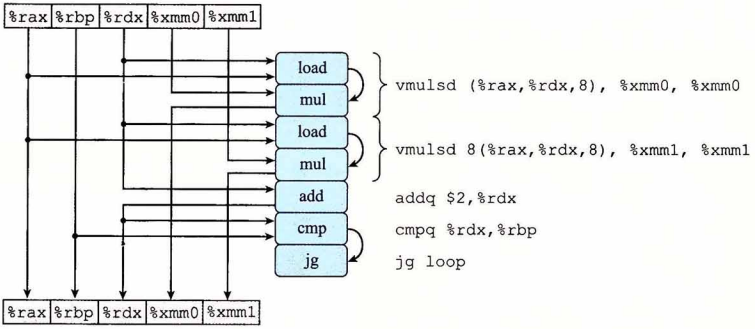

图 5-22　combine6 内循环代码的图形化表示。每次循环有两条 vmulsd 指令，每条指令被翻译成一个 load 和一个 mul 操作

我们可以将多个累积变量变换归纳为将循环展开 k 次，以及并行累积 k 个值，得到 $k\times k$ 循环展开。图 5-25 显示了当数值达到 $k=10$ 时，应用这种变换的效果。可以看到，当 k 值足够大时，程序在所有情况下几乎都能达到吞吐量界限。整数加在 $k=7$ 时达到的 CPE 为 0.54，接近由两个加载单元导致的吞吐量界限 0.50。整数乘和浮点加在 $k\geqslant 3$ 时达到的 CPE 为 1.01，接近由它们的功能单元设置的吞吐量界限 1.00。浮点乘在 $k\geqslant 10$ 时达到的 CPE 为 0.51，接近由两个浮点乘法器和两个加载单元设置的吞吐量界限 0.50。值得注意的是，即使乘法是更加复杂的操作，我们的代码在浮点乘上达到的吞吐量几乎是浮点加可以达到的两倍。

通常，只有保持能够执行该操作的所有功能单元的流水线都是满的，程序才能达到这个操作的吞吐量界限。对延迟为 L，容量为 C 的操作而言，这就要求循环展开因子 $k\geqslant C\cdot L$。比如，浮点乘有 $C=2$，$L=5$，循环展开因子就必须为 $k\geqslant 10$。浮点加有 $C=1$，$L=3$，则在 $k\geqslant 3$ 时达到最大吞吐量。

在执行 $k\times k$ 循环展开变换时，我们必须考虑是否要保留原始函数的功能。在第 2 章已经看到，补码运算是可交换和可结合的，甚至是当溢出时也是如此。因此，对于整数数据类型，在所有可能的情况下，combine6 计算出的结果都和 combine5 计算出的相同。因此，优化编译器潜在地能够将 combine4 中所示的代码首先转换成 combine5 的二路循环展开的版本，然后再通过引入并行性，将之转换成 combine6 的版本。有些编译器可以做这种或与之类似的变换来提高整数数据的性能。