5.8　这道题目说明了程序中小小的改动可能会造成很大的性能不同，特别是在乱序执行的机器上。图 5-39 画出了该函数一次迭代的 3 个乘法操作。在这张图中，关键路径上的操作用黑色方框表示——它们需要按照顺序计算，计算出循环变量 r 的新值。浅色方框表示的操作可以与关键路径操作并行地计算。对于一个关键路径上有 P 个操作的循环，每次迭代需要最少 $5P$ 个时钟周期，会计算出 3 个元素的乘积，得到 CPE 的下界 $5P/3$。也就是说，A1 的下界为 5.00，A2 和 A5 的为 3.33，而 A3 和 A4 的为 1.67。我们在 Intel Core i7 Haswell 处理器上运行这些函数，发现得到的 CPE 值与前述一致。

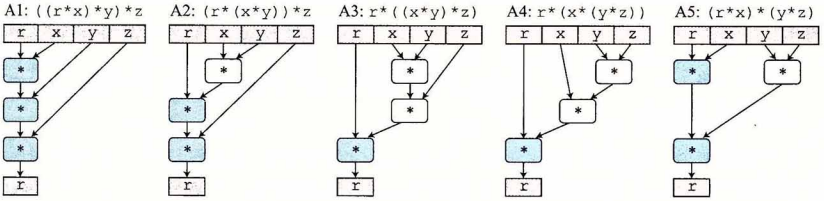

图 5-39　对于练习题 5.8 中各种情况乘法操作之间的数据相关。用黑色方框表示的操作形成了迭代的关键路径

5.9　这道题又说明了编码风格上的小变化能够让编译器更容易地察觉到使用条件传送的机会：

```
while (i1 < n && i2 < n) {
    long v1 = src1[i1];
    long v2 = src2[i2];
    long take1 = v1 < v2;
    dest[id++] = take1 ? v1 : v2;
    i1 += take1;
    i2 += (1-take1);
}
```

对于这个版本的代码，我们测量到 CPE 大约为 12.0，比原始的 CPE 15.0 有了明显的提高。

5.10　这道题要求你分析一个程序中潜在的加载-存储相互影响。

A. 对于 $0 \leqslant i \leqslant 998$，它要将每个元素 a[$i$]设置为 $i+1$。

B. 对于 $1 \leqslant i \leqslant 999$，它要将每个元素 a[$i$]设置为 0。

C. 在第二种情况中，每次迭代的加载都依赖于前一次迭代的存储结果。因此，在连续的迭代之间有写/读相关。

D. 得到的 CPE 等于 1.2，与示例 A 的相同，这是因为存储和后续的加载之间没有相关。

5.11　我们可以看到，这个函数在连续的迭代之间有写/读相关——一次迭代中的目的值 p[i]与下一次迭代中的源值 p[i- 1]相同。因此，每次迭代形成的关键路径就包括：一次存储(来自前一次迭代)，一次加载和一次浮点加。当存在数据相关时，测量得到的 CPE 值为 9.0，与 write_read 的 CPE 测量值 7.3 是一致的，因为 write_read 包括一个整数加(1 时钟周期延迟)，而 psum1 包括一个浮点加(3 时钟周期延迟)。

5.12　下面是对这个函数的一个修改版本：

```
1   void psum1a(float a[], float p[], long n)
2   {
3       long i;
4       /* last_val holds p[i-1]; val holds p[i] */
5       float last_val, val;
6       last_val = p[0] = a[0];
7       for (i = 1; i < n; i++) {
8           val  = last_val + a[i];
9           p[i] = val;
10          last_val = val;
11      }
12  }
```