和 B 的 n^2 个元素中的每一个都要读 n 次。计算 C 的 n^2 个元素中的每一个都要对 n 个值求和。不过，如果分析最里层循环迭代的行为，我们发现在访问数量和局部性上还是有区别的。为了分析，我们做了如下假设：

- 每个数组都是一个 double 类型的 $n \times n$ 的数组，sizeof(double)==8。
- 只有一个高速缓存，其块大小为 32 字节($B=32$)。
- 数组大小 n 很大，以至于矩阵的一行都不能完全装进 L1 高速缓存中。
- 编译器将局部变量存储到寄存器中，因此循环内对局部变量的引用不需要任何加载或存储指令。

code/mem/matmult/mm.c

```
1  for (i = 0; i < n; i++)
2      for (j = 0; j < n; j++) {
3          sum = 0.0;
4          for (k = 0; k < n; k++)
5              sum += A[i][k]*B[k][j];
6          C[i][j] += sum;
7      }
```

code/mem/matmult/mm.c

a) *ijk*版本

code/mem/matmult/mm.c

```
1  for (j = 0; j < n; j++)
2      for (k = 0; k < n; k++) {
3          r = B[k][j];
4          for (i = 0; i < n; i++)
5              C[i][j] += A[i][k]*r;
6      }
```

code/mem/matmult/mm.c

c) *jki*版本

code/mem/matmult/mm.c

```
1  for (k = 0; k < n; k++)
2      for (i = 0; i < n; i++) {
3          r = A[i][k];
4          for (j = 0; j < n; j++)
5              C[i][j] += r*B[k][j];
6      }
```

code/mem/matmult/mm.c

e) *kij*版本

code/mem/matmult/mm.c

```
1  for (j = 0; j < n; j++)
2      for (i = 0; i < n; i++) {
3          sum = 0.0;
4          for (k = 0; k < n; k++)
5              sum += A[i][k]*B[k][j];
6          C[i][j] += sum;
7      }
```

code/mem/matmult/mm.c

b) *jik*版本

code/mem/matmult/mm.c

```
1  for (k = 0; k < n; k++)
2      for (j = 0; j < n; j++) {
3          r = B[k][j];
4          for (i = 0; i < n; i++)
5              C[i][j] += A[i][k]*r;
6      }
```

code/mem/matmult/mm.c

d) *kji*版本

code/mem/matmult/mm.c

```
1  for (i = 0; i < n; i++)
2      for (k = 0; k < n; k++) {
3          r = A[i][k];
4          for (j = 0; j < n; j++)
5              C[i][j] += A[i][k]*r;
6      }
```

code/mem/matmult/mm.c

f) *ikj*版本

图 6-44　矩阵乘法的六个版本。每个版本都以它循环的顺序来唯一地标识

图 6-45 总结了我们对内循环的分析结果。注意 6 个版本成对地形成了 3 个等价类，用内循环中访问的矩阵对来表示每个类。例如，版本 *ijk* 和 *jik* 是类 AB 的成员，因为它们在最内层的循环中引用的是矩阵 A 和 B(而不是 C)。对于每个类，我们统计了每个内循环迭代中加载(读)和存储(写)的数量，每次循环迭代中对 A、B 和 C 的引用在高速缓存中不命中的数量，以及每次迭代缓存不命中的总数。

类 AB 例程的内循环(图 6-44a 和图 6－44b)以步长 1 扫描数组 A 的一行。因为每个高速缓存块保存四个 8 字节的字，A 的不命中率是每次迭代不命中 0.25 次。另一方面，内