订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

CS:APP2e Web Aside MEM:BLOCKING: 使用阻断来增加时间上的局部性*

Randal E. Bryant
David R. O'Hallaron

2012年6月5日

通知

本文档中的材料是《计算机系统，程序员的视角，第二版》*一书的补充材料，作者是Randal E. Bryant和David R. O'Hallaron，由Prentice-Hall出版，版权为2011年。在本文件中，所有以"CS:APP2e "开头的引用都是指这本书。关于这本书的更多信息可在*`csapp.cs.cmu.edu`*上找到。*

本文件向公众提供，但须遵守版权规定。你可以自由地复制和分发，但你不应该在没有注明出处的情况下使用这些材料的任何内容。

1 简介

有一种有趣的技术叫做*阻塞*，可以改善内循环的时间定位。阻塞的一般概念是将程序中的数据结构组织成大块，称为*块*。(在这里，"块 "指的是应用层的数据块，而*不是*缓存块）。程序的结构是这样的：它将一个块加载到L1高速缓存中，对该块进行所有需要的读写，然后丢弃该块，加载下一个块，如此循环。

与改善空间定位的简单循环转换不同，阻塞使代码更难阅读和理解。由于这个原因，它最适合用于优化编译器或经常执行的库例程。尽管如此，这项技术仍然值得研究和理解，因为它是一个普遍的概念，可以在一些系统上产生很大的性能提升。

*Copyright ©2010, R. E. Bryant, D. R. O'Hallaron.保留所有权利。

2 矩阵乘法的阻塞版本

屏蔽矩阵乘法程序的工作原理是将矩阵划分为子矩阵，然后利用这些子矩阵可以像标量一样被操作的数学事实。例如，假设我们想计算$C=AB$，其中A、B和C都是8×8的矩阵。那么我们可以将每个矩阵划分为四个4×4的子矩阵。

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \quad \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

其中

$$\begin{array}{ll} C_{11} & A_{11}B_{11} + A_{12}B_{21} \\ C_{12} & a_{11}b_{12} + a_{12}b_{22} \\ C_{21} & A_{21}B_{11} + A_{22}B_{21} \\ C_{22} & a_{21}b_{12} + a_{22}b_{22} \end{array}$$

图1显示了阻断矩阵乘法的一个版本，我们称之为*bijk*版本。这段代码的基本思想是将A和C划分为$1\times b$*大小的行片段*，将B划分为b*大小*$\times b$*大小的*块。最里面的(j, k)循环对将A的一个分片乘以B的一个块，并将结果累积到C的一个分片。

图2给出了图1中被封锁的代码的图形解释。关键的想法是，它将B的一个块加载到高速缓存中，用完后再丢弃它。对A的引用享有良好的空间定位，因为每个分片的访问跨度为1，还有良好的时间定位，因为整个分片被连续引用了*bsize*次。对B的引用具有良好的时间定位性，因为整个*bsize*×*bsize*块被连续访问了n次。最后，对C的引用具有良好的空间位置性，因为条形图的每个元素都是连续写入的。请注意，对C的引用没有良好的时间定位性，因为每个分块只被访问一次。

分块可以使代码更难读，但它也可以带来巨大的性能红利。图3显示了在Pentium III Xeon系统（*bsize*=25）上两个版本的阻塞式矩阵乘法的性能。请注意，与最好的非阻塞式版本相比，阻塞式版本的运行时间提高了2倍，从每次迭代约20个周期降低到每次迭代约10个周期。关于阻塞的另一个有趣的事情是，随着数组大小的增加，每次迭代的时间几乎保持不变。对于小的数组大小，阻断版本的额外开销导致其运行速度比非阻断版本慢。在n=100处有一个交叉点，在这之后，阻塞版本运行得更快。

我们要提醒的是，屏蔽矩阵乘法并不能提高所有系统的性能。例如，在Core i7系统上，存在未屏蔽的矩阵乘法版本，其性能与最佳屏蔽版本相同。

code/mem/matmult/bmm.c

```
void bijk(array A, array B, array C, int n, int bsize)
{
    int i, j, k, kk, jj;
    双和。
    int en = bsize * (n/bsize); /*均匀地放入块中的数量 */

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            C[i][j] = 0.0。

    for (kk = 0; kk < en; kk += bsize) {
        for (jj = 0; jj < en; jj += bsize) {
            for (i = 0; i < n; i++) {
                for (j = jj; j < jj + bsize; j++) {
                    sum = C[i][j];
                    for (k = kk; k < kk + bsize; k++) {
                        sum += A[i][k]*B[k][j]。
                    }
                    C[i][j] = sum;
                }
            }
        }
    }
}
```

code/mem/matmult/bmm.c

图1：**分块矩阵乘法。**一个简单的版本，假设阵列大小（n）是块大小（bsize）的整数倍。

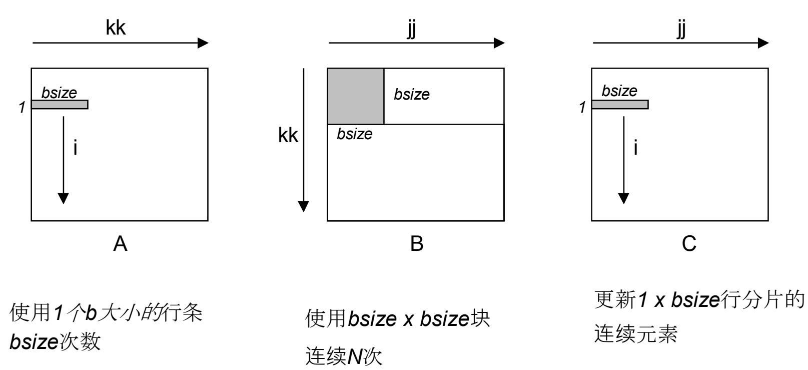

图2：**封锁式矩阵乘法的图形解释** 最内层的（j，k）循环对乘以一个 1×*bsize的*A的片断由*bsize*×*bsize的*B的块组成，并累积为1×*bsize的*C的片断。

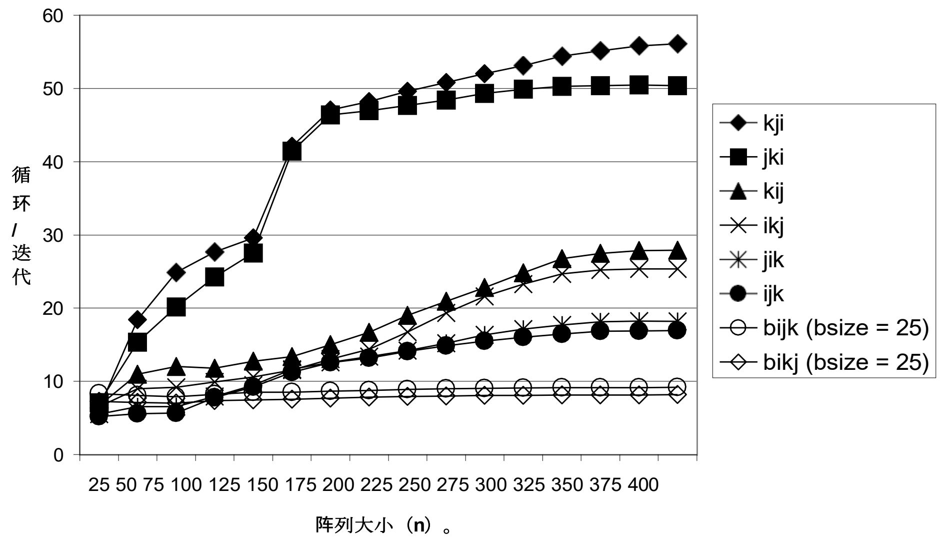

图3：**Pentium III Xeon封锁式矩阵乘法的性能。**图例：*bijk*和*bikj*：两个不同版本的阻塞式矩阵乘法。不同的非阻塞版本的性能显示出来供参考。