CS: APP3e Web Aside MEM: BLOCKING: 屏蔽: 使用阻塞来增加时间局部性

Randall e. Bryant
大卫 · r · 奥哈拉伦

March 24,20152015 年 3 月 24 日

公告

本文件中的材料是《计算机系统，程序员的视角》第三版的补充材料，作者是 Randal e. Bryant 和 David r. o'hallaron，由 Prentice-Hall 出版，2016 年版权所有。在这份文件中，所有以“ CS: APP3e”开头的参考文献都参考了这本书。更多关于这本书的信息请访问 csapp.cs.cmu.edu。

根据版权条款，本文档将向公众开放。你可以自由地复制和分发它，但是你必须为任何使用这些材料提供归属。

引言

有一种叫做阻塞的有趣技术可以改善内部循环的时间局部性。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。(在这种情况下，“块”指的是应用程序级别的数据块，而不是缓存块程序的结构是这样的: 它将一个数据块加载到 l1 缓存中，对该数据块执行所有需要的读写操作，然后丢弃该数据块，加载到下一个数据块，等等。

与改善空间局部性的简单循环转换不同，阻塞使代码更难阅读和理解。出于这个原因，它最适合优化编译器或频繁执行的库例程。尽管如此，这项技术仍然是值得研究和理解的，因为它是一个通用的概念，可以在一些系统上产生巨大的性能提升。

矩阵乘法的阻塞版本

阻塞矩阵乘法例程通过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操作的数学事实来工作。例如，假设我们想计算 c = AB，其中 a，b，c 各为 8 个矩阵。然后我们可以把每个矩阵分成 4 个子矩阵:

$$\begin{matrix} C21 & C22 \\ C11 & C12 \end{matrix} = \begin{matrix} 答 21 & 答 22 \\ 答 十 一 & 答 十 二 \\ 11 & 二 \end{matrix} \quad \begin{matrix} B21 & B22 \\ 乙 & 十 \\ B11 & 二 \end{matrix}$$

在哪儿

$$C1_1 = A11B11 + A12B21$$

$$C1_2 = A11B12 + A12B22$$

$$C2_1 = A21B11 + A22B21$$

$$C2_2 = A21B12 + A22B22$$

图 1 显示了阻塞矩阵乘法的一个版本，我们称之为 bijk 版本。这段代码背后的基本思想是将 a 和 c 划分为 1 个 bsize 行片段，并将 b 划分为 bsize bsize 块。最里面的(j; k)循环对将 a 的一个子块乘以 b 的一个子块，然后将结果累积为 c 的一个子块。I 循环迭代 a 和 c 的 n 行片，使用 b 中的相同块。

图 2 给出了图 1 中阻塞代码的图形化解释。关键的想法是，它加载一个 b 块到缓存中，使用它，然后丢弃它。对 a 的引用具有良好的空间局部性，因为每个银片的访问步长为 1。还有很好的时间局部性，因为整个银片是连续的 bsize 时间参考。对 b 的引用具有良好的时间局部性，因为整个 bsize bsize 块连续 n 次被访问。最后，c 的引用具有良好的空间局部性，因为银条的每个元素都是连续写入的。注意，对 c 的引用不具有良好的时间局部性，因为每个银条只被访问一次。

阻塞会使代码更难读取，但也会带来巨大的性能红利。图 3 显示了两个版本的阻塞矩阵在 Pentium III Xeon 系统(bsize = 25)上的性能。注意，阻塞使运行时间比最好的非阻塞版本提高了 2 倍，从每次迭代大约 20 个周期减少到每次迭代大约 10 个周期。关于阻塞的另一个有趣的事情是，每次迭代的时间几乎随着数组大小的增加而保持不变。对于较小的数组大小，阻塞版本的额外开销导致其运行速度比非阻塞版本慢。在 n = 100 处有一个交叉点，之后阻塞版本运行得更快。

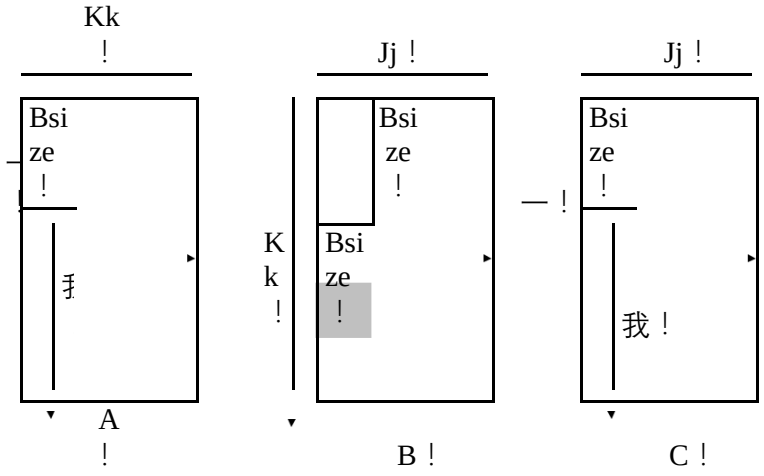
我们警告说，阻塞矩阵相乘并不能改善所有系统的性能。例如，在现代的 Core i7 系统中，存在未阻塞版本的矩阵乘法，其性能与最好的阻塞版本相同。

代码/mem/matmult/bmm.c

```
1 void bijk (数组 a，数组 b，数组 c，int n，int bsize)
2 {
Int i，j，k，kk，jj;
四重和;
5int en = bsize * (n/bsize) ;/* 与块相等的数量 */
6
(i = 0; i < n; i + +)
(j = 0; j < n; j + +)
9 c [ i ][ j ] = 0.0;
10
(kk = 0; kk < en; kk + = bsize){
(jj = 0; jj < en; jj + = bsize)
13                  (i = 0;          I < n; i + +){
14                      (j = jj; j < jj + bsize; j + +){
                            总
15                          和   = c [ i ][ j ] ;
16                          为   (k = kk; k < kk + bsize; k + +){
17                                   总和 + = a [ i ][ k ] * b [ k ][ j ] ;
18                          }
19                          C [ i ][ j ] = 和;
20                      }
21                  }
22              }
23          }
24 }
```

代码/mem/matmult/bmm.c

图 1: 阻塞矩阵相乘。一个简单的版本，假设数组大小(n)是块大小(bsize)的整数倍。



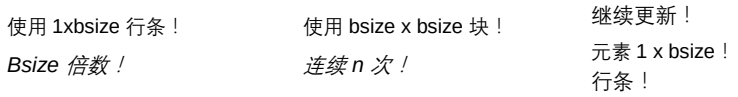


图 2: 分块矩阵的图形解释乘以最内层的(j; k)循环对乘以 a 的 1bsize 块乘以 b 的 1bsize 块，并累积成 c 的 1bsize 块。

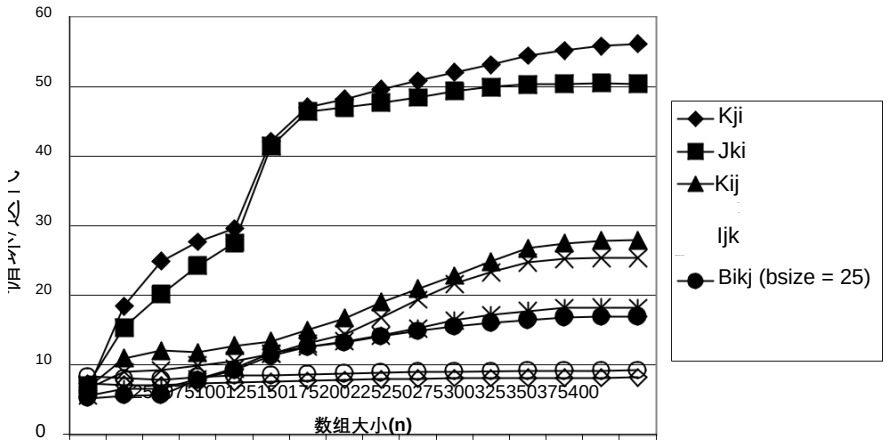


图 3: Pentium III Xeon 阻塞矩阵乘法性能。图例: bijk 和 bikj: 两种不同版本的阻塞矩阵相乘。不同的未阻塞版本的性能显示为参考。