国 防 科 技 大 学 学 报
第21卷 第3期 JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY Vol.21 No.3 1999

主维对矩阵运算性能的影响*

窦朝晖 胡庆丰 张秀山

(国防科技大学计算机系 长沙 410073)

摘 要 随着计算机体系结构的发展,高速缓存(cache)的引入,分块方法成为矩阵计算中性能优化的主要方法,而矩阵主维对分块算法的性能影响很大。本文分析了矩阵主维影响性能的原因以及如何选取主维来改善性能,并与拷贝方法进行了比较。最后用矩阵乘法和 LU 分解进行了试算,取得了满意的结果。

关键词 矩阵主维,分块,高速缓存,算法,优化,性能

分类号 TP301

The Effect of Leading Dimension to Performance of Matrix Computation

Dou Zhaohui Hu Qingfeng Zhang Xiushan

(Department of Computer Science, NUDT, Changsha, 410073)

Abstract With the development of computer architecture and the introduction of cache, blocking has been the main method to optimize performance in matrix computing, and the effect of leading dimension becomes important to blocking algorithms' performance. This paper analyzes this effect and how to select leading dimension to improve perofrmance, and compares this method with copy method. This method is applied to matrix multiplication and LU factorization, and the practical results agree with the theoretical analysis.

Key words leading dimension, blocking, cache, algorithm, optimization, performance

随着计算机体系结构的发展,高速缓存(cache)的引入,如何提高计算问题访存局部性,充分发挥 cache 的作用,尽量降低 cache 失效率对于科学计算应用具有重要意义。

分块方法是提高矩阵计算访存局部性的主要方法。但由于数组大小远超过 cache 的容量,对于相联度较小,特别是直接映射的 cache,当矩阵的阶为2的幂时,分块后块矩阵映射到 cache 时自冲突现象十分突出,严重影响了 cache 性能的发挥,此时分块算法的性能并未得到多大提高。由于分块算法引入了额外开销,有时分块算法的性能比不分块还要差。本文针对矩阵计算在 FORTRAN 语言程序设计中引入的矩阵主维,深入分析其对矩阵分块算法性能的影响,提出了简单实用的加边法经验公式。

1 矩阵主维影响性能的原因

矩阵主维对性能的影响,其实质可归结到 *cache* 对性能的影响。为了下面讨论方便,我们假设所用计算机系统的 *cache* 容量为 C,行长度为 l,相联度为1(即直接映射),以 LRU 方式工作。

设 $\boldsymbol{A}$ 是主维为 lda 的二维矩阵,A^* 为 $\boldsymbol{A}$ 的基地址,则 $A(j_1,j_2)$ 和 $A(j_1',j_2')$ 在主存中的地址分别为 $A^* + (j_2 - 1)lda + j_1 - 1$ 和 $A^* + (j_2' - 1)lda + j_1' - 1$,其地址差为 $d = (j_2' - j_2)lda + j_1' - j_1 (j_2' > j_2)$。那么它们在 cache 中的地址差则为 $cd = d \bmod C$。若 $cd = 0$ 时这两个元素就被映射到 cache 的同一单元。当矩阵规模很大时,即使在分块算法中,同一子块中的元素在 cache 中的地址差 cd 也可能为0,从而块矩阵也存在自冲突。

从 cd 的表达式中可以看出 cd 与矩阵主维 lda 有关,所以适当地选取 lda,可改变数组元素在 cache 中的地址差,使原来映射到同一 cache 单元的元素位于 cache 的不同单元,从而大大降低 cache 的自冲突,提高命中率,使性能得到提高。这对于分块算法效果尤为明显,因为分块算法消除了 cache 容量失

* 1998年10月26日收稿
第一作者:窦朝晖,男,1968年生,硕士生

效,再通过主维的变化来消除cache 自冲突,使cache 能得到充分利用,使矩阵运算的性能得到进一步提高。改变 lda,实际上就是给矩阵加边,所以在文献[1]中把这种方法称为加边法。

2 加边法与拷贝法的比较

在文献[2],[3],[4]中介绍了提高 cache 性能的拷贝方法。从实验结果看,拷贝方法比加边的效果好。从理论上分析,拷贝方法中子矩阵各元素是连续的,在分块不太大时能完全消除自冲突。在文献[1]给出了如何加边的定理,在分块大小较小时,根据该定理来选取矩阵主维可望获得较高的性能。但目前计算机系统 cache 容量较大,此时分块算法中分块要较大时,才能获得较高的性能。在运算规模很大时,用该定理选主维所需的额外存储空间太大,在理论上消除自冲突的同时,可能引入了交叉冲突。而拷贝法所需的额外存储空间只与分块大小有关,与问题规模无关,在同一时间所要访存的矩阵规模都很小,且可使这些矩阵在主存的地址是连续的,从而可消除交叉冲突。

由于 cache 的容量一般为2的方幂,由 cd 的表达式可知,当主维为奇数时,矩阵各元素发生自冲突的可能性大大减少。再考虑到前面分析的影响性能的制约因素,得出最简单的选取主维 lda 的经验公式:

$$lda = \begin{cases} n+1 & \text{当为偶数时} \\ n & \text{当为奇数时} \end{cases}$$

其中 n 为矩阵第一维大小。按上述公式选取主维,实现起来非常简单,且所需的额外存储空间已降低到最小,而实践证明这种方法对性能优化较为有效,所以上述经验公式在实际应用中是非常有用的。

另外,拷贝方法一般适用于库程序,而加边法一般适用于具体的应用程序。当然在具体应用程序开发中联合使用两种方法,效果更好。

3 分块 LU 分解的优化

假设 $\boldsymbol{A}$ 是 n 阶矩阵,并划分 $\boldsymbol{A}$ 成一个 $l \times l$(为简单起见,不妨假设 $n = ml$)的分块矩阵如下:

$$\boldsymbol{A} = (\boldsymbol{A}_0\ \boldsymbol{A}_1\ \cdots\ \boldsymbol{A}_1) \tag{1}$$

其中 $\boldsymbol{A}_i, i = 0, \cdots, l-1$ 是 $n \times m$ 矩阵,通过分析下式得到分块 LU 分解的原理:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{L}_{00} & \\ \boldsymbol{L}_{10} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{00} & \tilde{\boldsymbol{A}}_{01} \\ & \tilde{\boldsymbol{A}}_{11} \end{pmatrix} \tag{2}$$

其中 $\boldsymbol{L}_{00}$ 是 m 阶单位下三角矩阵,$\boldsymbol{U}_{00}$ 是 m 阶上三角矩阵,$\boldsymbol{L}_{10}$,$\tilde{\boldsymbol{A}}_{01}$ 和 $\tilde{\boldsymbol{A}}_{11}$ 分别是 $(n-m) \times m$,$m \times (n-m)$,和 $(n-m) \times (n-m)$ 矩阵,如果记

$$(\boldsymbol{A}_1 \quad \cdots \quad \boldsymbol{A}_{l-1}) = \begin{pmatrix} \boldsymbol{A}_{01} \\ \boldsymbol{A}_{11} \end{pmatrix}$$

其中 $\boldsymbol{A}_{01}$ 和 $\boldsymbol{A}_{01}$ 分别是 $m \times (n-m)$,和 $(n-m) \times (n-m)$ 矩阵,通过比较(2)式的两边得到:

$$\boldsymbol{A}_0 = \begin{pmatrix} \boldsymbol{L}_{00} \\ \boldsymbol{L}_{10} \end{pmatrix} \boldsymbol{U}_{00}, \boldsymbol{A}_{01} = \boldsymbol{L}_{00}\tilde{\boldsymbol{A}}_{01}, \boldsymbol{A}_{11} = \boldsymbol{L}_{10}\tilde{\boldsymbol{A}}_{01} + \tilde{\boldsymbol{A}}_{11} \tag{3}$$

可以看出,$\boldsymbol{L}_{00}$,$\boldsymbol{L}_{10}$ 和 $\boldsymbol{U}_{00}$ 是子矩阵 $\boldsymbol{A}_0$ 的 LU 分解,所以 $\tilde{\boldsymbol{A}}_{01}$ 和 $\tilde{\boldsymbol{A}}_{11}$ 可由(3)式计算得到,接下来的计算可对 $\tilde{\boldsymbol{A}}_{11}$ 重复上述过程,这就是分块 LU 分解方法。

从 LU 分解的特点知道,$\boldsymbol{A}_0$ 的 LU 分解和 $\tilde{\boldsymbol{A}}_{01}$ 的计算其数据重用率不高,而 $\tilde{\boldsymbol{A}}_{11}$ 的计算数据重用率很高,因此要对分块 LU 分解进行优化,着重应对 $\tilde{\boldsymbol{A}}_{11}$ 的计算进行优化。在实现过程中,矩阵 $\boldsymbol{A}_0$ 的 LU 分解我们是用 BLAS-2的 sscal 和 saxpy 实现的,$\tilde{\boldsymbol{A}}_{01}$ 是用 BLAS-3的 strsm 实现的。为了说明矩阵主维对性能的影响,$\tilde{\boldsymbol{A}}_{11}$ 的计算我们用两种方法实现:对矩阵分块且调用 BLAS-3的 sgemm 实现(记为方法1);调用带拷贝的分块矩阵乘(记为方法2)。我们对传统不分块 LU 分解(记为方法3)也进行了测试。

4 测试结果

我们在 DEC ALPHA 21164上进行了测试。该系统有三级 cache:一级 cache 分为指令 cache 和数据

cache, 均为容量8KB、直接映射、行长为32字节的 cache; 二级 cache 为指令和数据混合 cache, 容量为96KB, 采用3路组相联的映射方式, 行长为32或64字节; 三级 cache 是容量为1M、直接映射、行长为32或64字节的外部 cache。在这种 cache 结构下, 对分块矩阵乘, 我们发现按128×128的子块对矩阵 ***A***、***B***、***C*** 进行分块, 性能接近最优, 因此下面的测试结果都是基于这种划分, (1) 式中的 m 也为128。

文献[4]给出了矩阵乘的测试结果, 其中的方法2和方法3可归结为不同主维的分块矩阵乘, 这里不再赘述。

表1 是用方法1对不同阶的矩阵在不同主维下的试算结果。从中可看出, 矩阵主维对方法1的性能影响很大, 特别是把主维选为2的整幂(如本例中的2048)时, 性能最差, 而其它大小的主维, 性能差别不大, 而且性能在主维大小为奇数时比偶数时要好。因为我们的分块大小为128×128, 按文献[1]的定理, 似乎 *lda* 取2176(2176= 17×128)最好, 但实际结果并非如此。

表1中的各种组合我们对方法2和方法3也进行了测试, 性能随主维的变化规律基本一致, 限于篇幅只列出 n= 2048时的测试结果(见表2)。从表2可看出, 方法2最有效地解决了 cache 自冲突问题, 性能比较稳定, 受主维的影响不大。从表2还可看出传统方法受主维的影响不大, 这是因为此时的 cache 失效主要是由容量失效引起的, 自冲突不是此时的主要原因。

表1 方法1的测试结果

Table 1 The test result of method 1

运行时间(s) *lda* / *n*	2025	2040	2041	2048	2049	2056	2057	2176
2025	49.52	53.01	50.78	92.53	52.01	55.42	49.98	60.88
2040	–	53.76	51.44	94.13	52.69	56.12	51.10	62.01
2048	–	–	–	95.25	53.24	56.67	51.10	63.07
2056	–	–	–	–	–	57.70	52.08	63.74

表2 n= 2048的测试结果

Table 2 The test result of n= 2048

运行时间(s) *lda* / *n*	2048	2049	2056	2057	2176
方法1	95.25	53.24	56.67	51.10	63.07
方法2	40.25	37.73	38.70	37.10	38.93
方法3	105.99	94.37	96.10	92.97	99.38

5 结论

本文对加边和拷贝两种 cache 优化方法进行了比较, 提出了使性能接近最优的、简单实用的主维经验公式。加边法的效果不如拷贝方法, 但它实现简单, 且应用层次不同, 故有一定的实用性。

参考文献

1 乔香珍. Cache 性能与程序优化. 计算机学报, 1996, 19(11): 819～823

2 Temam O, Granston E D, Jalby W. To copy or not to copy: a compile-time technique for accessing when data copying should be used to eliminate cache conflicts. In: Proc Supercomputing'93, 1993, 410～419

3 Lam M S, Rothberg E E, Wolf M E. The cache performance of blocked algorithms. In: Proc ASPLOS-IV, 1991, 63～74

4 窦朝晖, 胡庆丰. Copy- 提高分块算法性能的有效方法. 计算机工程与科学, 已录用