龙芯 2F 上的访存优化①

苏 波 李 凯 徐志广 何颂颂 (中国科学技术大学 计算机科学与技术系 安徽 合肥 230027)

摘 要: 一般的数据处理程序中，计算时间在其中往往只起次要作用，因此访存方式是否有效对程序的性能影响很大。在基于龙芯 2F 处理器研制的高性能计算机系统 KD-50-I 上安装 ATLAS，经测试其性能只达到龙芯 2F 理论峰值的 30%。通过循环展开减少函数存储访问次数，增大计算访存比；采用数据分块、部分拷贝以增强访存局部性，减少 cache 失效；利用非阻塞 cache 加快内存访问速度等访存优化技术，将 ATLAS 性能提高 50%以上。

关键词: ATLAS KD-50-I cache 失效 非阻塞 cache

Optimization of Memory Access Based on Loongson2F

SU Bo, LI Kai, XU Zhi-Guang, HE Song-Song

(School of Computer Science and Technology of USTC, Hefei 230027, China)

Abstract: In most cases, compared to computing time, memory access time takes a much larger proportion of program running time. Therefore, memory access approach can affect the program performance significantly. Testing results show that the performance of ATLAS transplanted on KD-50-I, which is based on Loongson 2F, reaches only 30% of its theoretical peak. In this paper, by exploiting Loop Unrolling technique to decrease memory access frequency, enhancing time and space locality to reduce cache misses and nonblocking cache mechanism to form memory access pipeline, the performance of optimized ATLAS can be improved to 50% higher.

Keywords: ATLAS; KD-50-I; cache miss; non-blocking cache

访存速度是抑制系统整体性能的瓶颈，而且一般数据处理算法很少能够有效地使用存储器的性能[1]。具有相同功能的程序在不同的存储访问方式下往往会出现迥异的运行速度，因此高效的访存方式对发挥系统和应用程序的性能至关重要。

1 前言

在基于龙芯 2F[2,3]处理器研制的高性能计算机系统 KD-50-I[4]上安装 BLAS[5,6](基本线性代数子程序库)的通用优化版本 ATLAS[5-7]，经测试其基本函数 dgemm 的运行速度只达到 800Mflops，sgemm 函数速度也仅在 1Gflops 左右，距龙芯 2F 的理论峰值 3Gflops 相差甚远。

本文在 ATLAS 的基础上针对龙芯 2F 体系结构特点，采用各种优化技术，特别是针对存储访问方式的优化，有效使用内存和 cache，实现了高效的 BLAS 库，各级函数性能相较 ATLAS 整体提升 50%以上。

2 龙芯2F处理器特征

龙芯 2F 处理器是一款实现 64 位 MIPS III 指令集的通用 RISC SOC 处理器，与程序优化密切相关的结构特征有[2]:

指令流水线每个时钟周期取四条指令译码，乱序执行(如寄存器重命名、转移预测和动态调度)、动态提交;

一级 cache 由 64KB 的指令 cache 和 64KB 的数据 cache 组成，二级 cache 大小为 512KB，均采用四路组相联的结构及随机替换算法;

TLB 有 64 项，采用全相联结构，每项可以映射

① 基金项目:国家高技术研究发展计划(863)(2008AA010902)
收稿时间:2009-03-04

一个奇页和一个偶页。通过 24 项的访存队列以及 8 项的访存失效队列来动态地解决地址依赖，实现访存操作的乱序执行、非阻塞 cache、取数指令猜测执行、写合并等优化技术。

32 个 64 位浮点寄存器,两个定点功能部件和两个浮点功能部件，支持乘加操作。

3 访存优化技术

程序的运行由访存和计算组成，若要加快程序的运算速度，不仅仅要分别减少访存时间和计算时间，还要对访存过程和计算过程进行合理的调度，使之更好地结合。BLAS 函数的计算量与访存量是同量级的，由于访存特别是访问内存的速度远远慢于处理器的计算速度，程序实际计算的时间在整个运行过程中处于次要地位，因此采用高效的访存方式能够有效地提程序的性能。

由于 CPU 计算部件和访存部件可以互相独立地工作，通过减少访存操作、高效使用 cache、加快内存访问等方法，使计算和访存能够几乎并行地进行，使得计算时间隐藏在访存之中，缩短程序运行时间。

3.1 减少访存次数

有效的访存优化方法要尽可能少地访问内存，在计算中应该适当地组织数据交换过程，对已取得的数据尽量地重用，避免不必要的数据访问。

在多重循环中展开外循环可以减少访存操作，提高计算访存比。

对于 BLAS 来说，其二级函数在计算过程中只需使用矩阵数据一遍，因此函数运行时主要访问内存。以 gemv 为例，假设程序主要循环的外层展开次数为 K，则访存次数为 $(K+1)N^2/K$，相比展开前的 $2N^2$ 次降低 $(K\text{-}1)/2K$，随 K 的增大而趋近于 1/2。

三级函数则主要是重复访问 cache,若在 gemm 的计算中将外两层循环按 $K_1\times K_2$ 方式进行展开，保证 cache 全命中(可以通过矩阵分块来实现，n 为分块大小)，则一次内循环一共需要读取 nK_1 个 A 值和 nK_2 个 B 值，以及 K_1K_2 个 C 值，共需访存 $n(K_1+K_2)+K_1K_2$ 次，如果不展开循环则需要访存 $n(K_1K_2+K_1)$ 次，降低约 $(K_1K_2-K_2)/(K_1K_2+K_1)$，计算量即乘加操作次数为 nK_1K_2，计算访存比约为 $K_1K_2/(K_1+K_2)$。

优化效率并不仅仅与循环展开次数有关，还需要考虑到处理器的具体结构。龙芯 2F 的浮点运算部件每周期平均可以完成两次浮点运算，而相同时间内处理器只能够从一级 cache 中读取一个 double 型数据，若是从内存中读取则更慢，因此需要考虑展开循环之后的计算访存比。过大的展开深度将导致循环尺寸急剧增加，而且由于浮点寄存器数目的限制，可能使得性能降低。经过测试，在龙芯 2F 上，二级双精度函数的外循环展开次数为 8 或者 16，而三级双精度函数外两层循环的展开层次为 4×4 时最好。图 1 是 dgemv 函数在 8×8 和 4×4 两种不同循环展开方式下的性能。

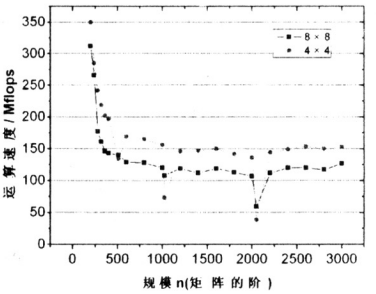

图 1　不同循环展开方式下 dgemv 性能

3.2 减少 cache 失效

cache 有效是由于程序对存储器的访问具有时间和空间上的局部性。如果 cache 容量不够或者程序访存的局部性不好，可能会导致 cache 失效的产生。cache 失效的情况分为首次失效、容量失效和冲突失效三种。对于 BLAS 函数来说，至少需要将矩阵或者向量装入一次 cache，首次失效无法避免；矩阵分块能减少容量失效；与冲突失效相关的因素有：cache 结构、矩阵规模和存储方式、数据的访问方式等，可以使用部分拷贝的方式来减少冲突失效。

3.2.1 矩阵分块

当矩阵的规模大于 cache 时，在计算中会使得先前装入 cache 的数据在重用前被替换出去。通过分块将矩阵按棋盘分成小块，每一小块需保证能全部装入 cache，并使得每一块数据在被装入 cache 后，都要在尽量充分重用之后才被替换掉。

分块后的算法性能和块的大小有很大关系，但是在 cache 容量允许的范围内．性能并不总是随着块的增大而提高。虽然可以通过拷贝的方法使得分块数据连续存放，但是由于龙芯 2F 的 cache 4 路组相连及随机替换策略，过大的分块仍然会出现 cache 冲突失效，因此在实际确定分块大小时，仍然需要进行一定

的测试。考虑到之前确定的循环展开因子，经过测试，在龙芯 2F 上，double 型矩阵的最佳分块方式为 60×60。在其他实现细节不变的情况下，将 dgemm 函数的分块与不分块的性能进行对比得到图 2，当矩阵的阶小于 200 时，两者性能几乎一致，随着 n 的增大，分块的优势逐渐体现出来，最终比不分块实现的性能提高了一倍以上。

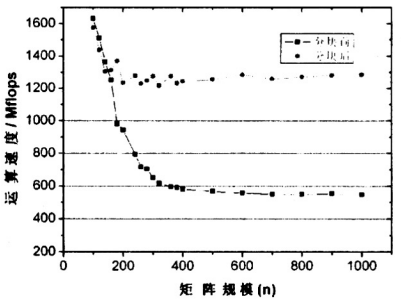

图 2　分块前后 dgemm 性能

3.2.2 部分拷贝

数据拷贝是将原本非连续存储的数据根据其访问顺序连续拷贝到另一个数组中。它使得计算中的数据访问连续，减少 cache 中的无用数据，并能减少冲突失效，提高性能。

在对数据子块进行运算之前，将其拷贝到一个数组之中连续存储，会明显提高 cache 命中率。以 gemm 为例，存在四种计算方式(T×T，T×N，N×N，N×T)，若是按列主序存储，则计算 T×N 时，对矩阵 A 和 B 的访问连续，此时的访存顺序较为有利。因此有目的地将参与子块按照计算方式进行转置拷贝，可以把四种形式的计算都转化成 T×N 形式，这样对两个小块都是按列顺序取数，既简化实现又可以提高效率。

龙芯 2F 的 cache 为 4 路组相连，1 级 cache 大小为 64K，若存取的数据地址相差为 16K(2048 个 double 数据)时，将会发生严重的 cache 冲突失效。

例如普通矩阵存储方式的二级函数(dgemv)热点代码中的访存部分：

```
A0 = A;          A1 = A0 +lda;
A2 = A1 +lda;    A3 = A2 +lda;
                 ...
```

A 为 double 型数组，外循环展开次数为 K，则位于任两行相同列的两个元素 A_{im} 与 A_{jm} 之间元素的个数为 $n=|i-j|\times LDA=kLDA,\ k<K$。当 $n=2048i$，也就是 $LDA=2048i/k(i\in N,\ k<K)$ 时，将 A_{jm} 调入后，很可能将 A_{im} 所处的 cache 行替换掉，从而发生冲突失效，需要从内存中再次读取，大大影响性能。K 与 i 的值越大，在 $LDA=2048i/k$ 时效率就越低(见图 1)。

如果将某一小块矩阵元素连续拷贝到另一个数组后，理论上当这个数组大小小于一路一级 cache 时，在新数组中将不存在可能发生冲突的数据，从而避免出现冲突失效。但 BLAS 二级函数运行时，矩阵数据原本就只需被读取一次，如果使用拷贝，会使得数据访问次数加倍，损失整体性能。但是对三级函数则没有这种限制，在矩阵乘法的过程中，会多次重复访问矩阵的元素，因此拷贝的开销相对来说可以容忍。

3.3 加快内存访问

龙芯 2F 处理器实现了非阻塞的 cache，即在产生失效时允许继续发射后续的访存指令，因此将会引起失效的访存指令连续发射，形成访存流水线，可以有效的重叠访存的延迟，加快程序运行速度。

以二级函数 sgpmv 为例介绍非阻塞 cache 机制在性能优化中的作用。如图 3 所示，压缩矩阵 A 实际上是存储在一个数组之中，矩阵第 i 行的起始地址为 $A+i\times LDA+i(i-1)/2$。

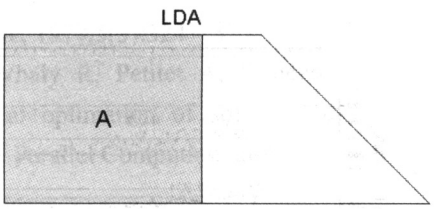

图 3　压缩矩阵 A 的存储方式(U)

Atlas 库对 sgpmv 的实现是

```
for( j = 0, jaj = 0, jy = 0; j < M; j++, jy +=
incy ) {
      t0 = 0;
    for( i = 0, iaij = jaj, ix = 0; i < N; i++,
  iaij += 1, ix += incx ) {
      t0 += A[iaij] * X[ix];
    }
    Y[jy] *= beta;
    Y[jy] += alpha * t0;
    jaj += lda; lda += 1;
}
```

该算法在 KD-50-I 上的运算速度约为 100M flops。将其按 8×8 循环展开得到算法 1。这里只给出算法 1 主要部分也就是上面算法标注部分的展开形式(表示展开循环后矩阵 A 对应第行的起始地址)：

```
x0 = X[ix];
t0 = A0[iaij]*x0;      t1 = A1[iaij]*x0;
t2 = A2[iaij]*x0;      t3 = A3[iaij]*x0;
(1)
t4 = A4[iaij]*x0;      t5 = A5[iaij]*x0;
t6 = A6[iaij]*x0;      t7 = A7[iaij]*x0;
    ...                    ...
x7 = X[ix+7*incx];
t0 = A0[iaij+7]*x7;    t1 = A1[iaij+7]*x7;
t2 = A2[iaij+7]*x7;    t3 =
```

算法 1 运算速度将近 200Mflops，提升约一倍，但仍不是非常理想。根据压缩矩阵 A 的存储方式分析这段代码，因为 $Ai-A(i-1)=LDA+i-1$，各行相同列的元素在 cache 行中所处位置不同。步骤(1)之后的每一步，在读取数据时，肯定会出现所需数据 cache 不命中的情况。

将步骤(8)提到(1)之前执行得到算法 2，两者性能如图 4 所示：

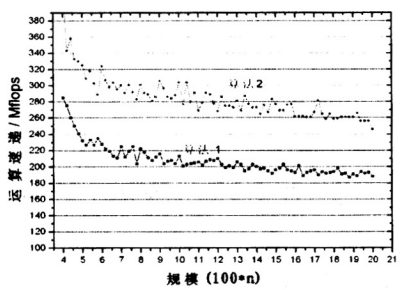

图 4　算法 1 和 2 性能比较

从图 4 可以看出算法 2 的运算速度大大加快，达到 250Mflops 以上。这是因为在运行(8)时，所读取的 8 个数据都会 cache 失效，因此处理器会连续发出 8 条内存访问请求。在步骤(8)之后的所有的操作中，所需读取的数据都已经处在 cache 之中，无需再访问内存。这样虽然没有减少内存访问，却将内存读取操作集中在了一起，充分利用龙芯 2F 的非阻塞 cache，形成内存访问流水线，提高了程序运行速度。

4　性能

在 BLAS 的三级函数中，一级函数的实现简单，ATLAS 库对其采用的优化已经达到了较好效果。对二级函数的优化主要体现在内存的有效使用，而三级函数则需要充分合理地利用 cache。

在下文中，假设矩阵 A 是连续存储的(LDA 等于 A 的第二维大小 N)。

4.1　二级函数

二级函数的计算过程中只需要一次访问矩阵，cache 的作用相对较小，容量失效一般不会出现，是否分块对函数性能影响不大，而部分拷贝会带来额外的访存开销，因此二级函数的优化主要体现在内存访问上。经过测试，在龙芯 2F 上，计算访内存比为 1：1 时(从内存读取一个 cache 行(8 个单精度数)同时实现 8 次乘加操作)的运算速度极限在 400Mflops 以上。

图 5 是二级库中的基本函数 sgemv 的优化过程，ATLAS 对函数主循环的 4×4 展开，相对于原始的 BLAS 库性能已有很大提升，然而距 400Mflops 仍然相差甚远。随着方阵规模 N 的变化，函数的性能会发生很大的周期性跳动，幅度几乎达到了 30%。改变展开方式为 8×8 后，运算速度达到 220Mflops，但剧烈波动仍然存在。调整内循环的指令次序，利用非阻塞 cache 流水访问内存，不但使速度到 270Mflops，同时削弱了性能波动的幅度。

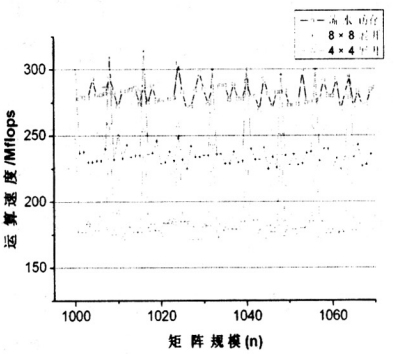

图 5　sgemv 优化过程

4.2　三级函数

三级库函数的优化主要体现在对 cache 的高效使用上，循环展开可以减少对 cache 的访问次数，分块和部分拷贝能够局部化访存，削弱 cache 失效对性能的影响。

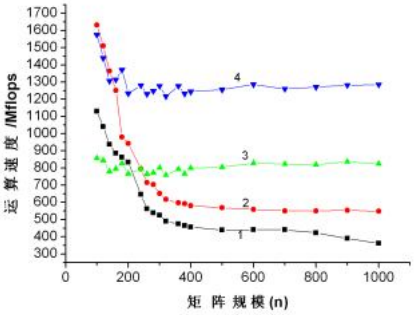

图 6 dgemm 优化过程

三级库主要函数 dgemm 的优化过程如图 6 所示。曲线 1 为函数在外层 5×3 展开未分块下的性能，外层 4×4 展开后得到曲线 2。将它们进行分块、数据拷贝之后得到曲线 3 和 4，分别是 ATLAS 实现和修改之后的性能，优化后的运算速度相较 ATLAS 提升了 50%以上。

表 1 三级库部分函数性能(Mflops)

	规模 200		规模 400		规模 800	
	ATLAS	优化后	ATLAS	优化后	ATLAS	优化后
dgemm	766	1234	799	1244	819	1271
dsymm	768	1262	795	1310	817	1301
dsyrk	642	1153	660	1082	691	1056
dtrmm	680	1142	723	1191	769	1242

三级库中其它各个函数实现方式很相近，很大部分都调用 gemm 作为其主体实现。

优化后的三级库双精度函数整体性能提升了 50%以上。

5 总结

ATLAS 的优化过程基本上是对存储器的高效使用：二级函数的优化主要是合理使用内存；三级函数则以 cache 访问为主要优化目标。在 KD-50-I 上优化 ATLAS 库，通过减少内存访问时间，提高 cache 访问局部性、消除 cache 失效，提升 BLAS 性能，对发挥龙芯 2F 处理器性能，加快科学计算速度有重要意义。

参考文献

1 Kaspersky K. 代码优化:有效使用内存.北京:电子工业出版社, 2004.85.

2 http://www.loongson.cn/loongson/

3 龙芯 2F 处理器用户手册.中国科学院计算技术研究所.

4 http://www.kd50.ustc.edu.cn

5 http://www.netlib.org

6 Lawson CL, Hanson RJ, et al. Basic linear algebra subprograms for Fortran usage. ACM Trans. Math. Software, 1979,5(3):324－325.

7 Clint Whaly R, Petitet A, Dongarra JJ. Automated empirical optimation of software and the ATL AS project. Parallel Computing, 2001,27(1-2):3－35.