稀疏矩阵向量乘的访存分析和优化

张秀霞[1+]，陈明宇[1]，李佳佳[1]，谭光明[1]

[1](中国科学院计算技术研究所，北 京 100190)

Memory Accessing Analysis of Sparse Matrix Vector Multiplication and Optimization

ZHANG Xiu-Xia[1+], CHEN Ming-Yu[1] ， LI Jia-Jia[1], TAN Guang-Ming[1]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: Phn +86-010-62600671, Fax +86-010-62600600, E-mail: zhangxiuxia@ict.ac.cn

Received 2011-07-30; Accepted 2011-09-15

Abstract: Sparse Matrix Vector multiplication (SpMV) is one of the most important kernel in computing science. Both theoretical analysis and practical experiments demonstrate that SpMV is a memory-intensive application. However state-of-the-art compilers cannot make full use of the modern processors' characters, the usage ratio of SpMV bandwidth is only about 10%. In this paper, based on the memory access characters of current processors, we optimize memory access in SpMV instruction pipeline using SIMD instruction. The experimental results show that there are 63% and 89% performance improvement on Intel Nehalem and Sandy Bridge, 30% and 36% improvement on AMD Opteron 6168 and Opteron 8374 compared with the standard SpMV implementation. As for matrices in real applications, there are 10% performance gained on Nehalem and Sandy Bridge.

Key words: sparse matrix vector multiplication; SpMV; pipeline; SIMD; Memory Access

摘 要：稀疏矩阵向量乘(SpMV)是科学计算中最重要的核心算法之一。理论分析和实际测试结果都表明，SpMV 属于访存密集型应用。由于目前主流编译器尚不能充分利用现代处理器的访存特性，SpMV 对带宽利用率仅为 10%。本文通过探索现代处理器的访存特征，使用单指令流多数据流（SIMD）对 SpMV 进行流水线访存优化。实验表明与标准 SpMV 相比，优化后的 SpMV 在 Intel Nehalem 和 Sandy Bridge 架构上流水线性能分别有 63%和 89%的提升，在 AMD Opteron 6168 和 Opteron 8374 HE 上分别有 30%和 36%的提升。SpMV 在实际矩阵进行的性能测试中，Intel Nehalem 和 Sandy Bridge 架构上均有 10%的性能提高。

关键字：稀疏矩阵向量乘; SpMV; 流水线; SIMD; 访存

中图法分类号：TP302 **文献标识码**: A

1 问题介绍

很多科学和工程问题，如物理模拟(粒子碰撞，核爆炸模拟)、医学成像，通过建模，最终可以转化成线性方程组求解问题。由于计算时间的要求，大规模线性方程组往往采用迭代法求解。迭代方法的核心操作是矩阵向量乘，并且这些线性方程组往往是稀疏的。因此对线性方程组求解的优化转化为稀疏矩阵向量乘(SpMV)的优化。

```
1    for (i=0; i<M; i++) {
2        t = 0.0;
3        for (j=row_start[i]; j<row_start[i+1]; j++) {
4            t += val[j] * x[col_idx[j]];
5        }
6        y[i]=t;
7    }
```

算法 1 SpMV 算法实现

稀疏矩阵的存储格式之一是 CSR，需要 row_start、

* Supported by National 863 Program under Grant No.2009AA01A129（国家 863 计划）; the National Natural Science Foundation of China under Grant No.60803030,61033009,60921002,60925009（国家自然科学基金）; the 973 Program under Grant No.2011CB302500（国家 973 计划）.

作者简介：张秀霞(1987－)，女，河南省柘城县人，硕士研究生，学生，主要研究领域为高性能计算；陈明宇(1972－)，男，博士，研究员，主要研究领域为计算机体系结构. 李佳佳(1988－)，女，博士研究生，学生，主要研究领域为并行计算；谭光明(1980－)，男，博士，副研究员，主要研究领域为并行计算

value、 col_idx 三个一维数组和行数(M)、非零元个数(NZ)来记录原始矩阵的信息。其中 row_start 记录每行第一个非零元素的起始位置，value 为按行存储的非零元素值，col_idx 为按行存储的非零元所在的列号。图 1 为 CSR 矩阵的 SpMV 操作的核心代码。

由于现在处理器的计算能力和 RAM 的容量增长远远超过对 RAM 访问速度的增长，CPU 和 RAM 的速度的剪刀差越来越大。SpMV 属于访存密集型程序，如果没有足够的带宽，SpMV 的性能也难以得到根本性提高。目前对 SpMV 对访存带宽利用率仅为 10%[1]。由于其间接访存和访存的不规则性，增加了编译器对其优化的难度。通过反汇编，我们发现目前的主流编译器 gcc、icc、Open64 均未使用 128-bit 的 SSE 访存指令对其进行 SIMD 向量优化。因此对于双精度 SpMV 操作访存带宽损失了一倍。本文考虑到访存和计算的差距，编译器和处理器的发展的不同步性，针对主流处理器 Intel 的 Nehalem、Sandy Bridge 和 AMD 的 Opteron 6168、Opteron 8374 HE 进行了汇编级的 SIMD 优化，充分利用处理器的访存带宽。

2 SpMV 相关工作

串行 SpMV 的优化主要分为两个方向：一是数据压缩的方法，通过减少数据存储从而减少数据对内存的访问，如 delta 编码、csx 编码；另一类是基于体系结构的优化，通过数据局部性减少访存延迟，如寄存器分块、cache 分块、TLB 分块等。这些优化都在高级语言级进行（如 C 语言级），作者目前尚未看到对 SpMV 进行汇编级优化的相关文献。

3 SpMV 优化及实验结果

3.1 SpMV流水线分析与优化

从算法 1 SpMV 的实现代码可以看出，对于双精度矩阵其访存大小为(NZ*8+NZ*8+NZ*4+M*2*4)字节，浮点计算的次数为 NZ*2。从代码 line 4 可以看出每 3 次访存做 2 次浮点运算。对于 Intel Nehalem 和 Westmere 处理器，由于只有一个读端口，一次至多发射一条 Load 指令。而处理器的计算端口相对较多（浮点加和浮点乘分别占用不同的端口），如果没有数据依赖，一个 cycle 可以同时发射 2 个浮点计算(浮点加和乘)。由此可见，SpMV 是典型的访存密集型应用，这就造成计算等待访存数据的局面，也就是 SpMV 性能瓶颈所在。因此对其带宽进行分析的重要性要高于对其浮点效率的分析。

带宽:单位时间内传输的数据量。

$$\text{Bandwidth} = \frac{\text{传输的数据量}}{\text{传输时间}(\text{数据量单位 byte，传输时间单位时钟周期})} \qquad \text{公式 1}$$

带宽利用率:所关注的带宽与峰值带宽的利用率

$$\text{Bandwidth Efficency(BE)} = \frac{\text{Bandwidth}}{\text{峰值带宽}} \qquad \text{公式 2}$$

表 1 SpMV 的访存情况

访问的数据	*类型：次数*	*总大小 (byte)*	*原始访问粒度 (byte)*	*SSE 优化后访问粒度(Nehalem)*	*SSE 优化后访问粒度 (Sandy Bridge)*	*访 问 类型*
矩阵 A	Double: NZ	Nz*8	8	16	32	只读
col_idx	Int: NZ	Nz*4	4	8，16	32	只读
向量 x	Double:n	N*8	8	8	8	只读
向量 y	Double:m	M*8	8	8	8	只写

注:如果不展开，访问粒度为 8bytes，循环展开访问粒度为 16.

下面我们以 Intel Nehalem 架构为例，对 SpMV 程序带宽理论分析。结果公式 1、公式 2 及表 1 中的数据访问数据，可以计算出原始 SpMV 和使用 SSE 优化后程序的理论带宽峰值分别为 41.66% 和 71.4%。

由于我们主要针对 SpMV 的流水线优化，减少其他因素(如 Cache miss)的影响，所以对数据规模进行限制，保证使其都放入 L1 cache 中。另外考虑流水线调度、指令替换、冗余循环变量删除，调整指令顺序协助保留站工作编译优化策略。

3.2 流水线的优化结果

测试矩阵是来自实际矩阵中的截取一个片段，规模为 50x300, 2272 个非零元。矩阵 A 和向量 x，y 总大小为 30KB 左右(略小于 L1 data cache 大小)，下表为流水线优化结果。

表 2 Nehalem icc -xsse4.2 编译 VS 汇编

参数/版本	*STD*	*SSE*	*SSE-U2*	*SSE-U4*
时间（cycles）	7960	6815	5346	4873
带宽利用率	0.3569	0.4167	0.5312	0.5828
未利用带宽	0.3574	0.2976	0.1830	0.1314
加速比	1	1.17	1.49	1.63

表 3 Sandy Bridge icc -xsse4.2 编译 VS

参数/版本	*STD*	*SSE*	*SSE-U2*	*SSE-U4*
时间（cycles）	6536	4474	3664	3465
带宽利用率	0.2179	0.3173	0.3875	0.4098
未利用带宽	0.4969	0.3968	0.3267	0.3044
加速比	1	1.46	1.78	1.89

注：STD：SpMV 的标准 C 语言实现版，**SSE**： SpMV 的 SSE 实现 **SSE-U2**:SpMV SSE 实现循环展开 2 次 **SSE-U4**：SpMV SSE 实现循环展开 4 次

从表 2 和表 3 可以看出，通过 SSE 对访存进行优化，SpMV 的时间效率和带宽利用率有较大提高。就时间而言，单核 Nehalem 平台上有最高有 1.63 倍的加速比，Sandy Bridge 平台有 1.89 倍的加速比，带宽利用率接近理论值，从原来的从 35.69%提高到 58.28%。在 AMD Opteron 6168 平台上有 31%提高，Opteron 8374 有 36%提高。本程序通过汇编优化对流水线的优化是目前编译器无法做到的。为了寻找丢失的带宽的原因，我们使用 peformance counter 对程序进行分析。实验表明，未利用带宽一部分是由于分支预测失败引起的，另一部分是由于指令仍然存在一些相关，使得保留站有限而导致流水线停滞。

3.3 实测性能

为了检验本文所做优化在实际中的应用，我们对实际工程应用中的矩阵进行测试,我们选的矩阵来自多方面的应用，如金融、基于有限元的建模，电路模拟，网络爬虫等。这些矩阵也曾在 NVIDIA[2]和 William[3]的论文中使用。

通过统计分析，使用汇编优化比标准 SpMV 性能平均优优 10%，Sandy Bridge 上最高 20%，Nehelem 最高 40%的提高。对平均行元素比较多的矩阵提高比较多，填充比比较低，额外开销少。实验表明，通过 SIMD 指令对 SpMV 优化，可以增加访存的吞吐率，提高 SpMV 的性能。

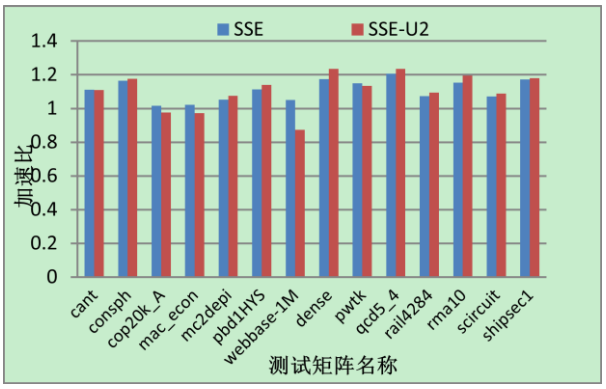

图 2 Sandy Bridge 上 SpMV 加速比

4 结论及展望

本文通过对 SpMV 核心算法进行以访存为主的流水线优化，较大提高了其流水线效率。优化后的 SpMV 流水线在 Intel Nehalem、 Sandy Bridge 和 AMD Opteron6168、Opteron 8374 HE 上分别有 63%，89%，31%，35%的性能提升。对实际矩阵，SpMV 的性能也有一定的提高：Nehalem、Sandy Bridge 上，均有 10%的性能提高。目前的优化主要集中于流水线级，将来可以进一步结合各级 Cache 和 DRAM 的优化方法，以充分利用现代处理器的各种资源。

References:

[1] Richard Vuduc, James Demmel, and Katherine Yelick. “OSKI: A library of automatically tuned sparse matrix kernels” Proceedings of SciDAC 2005, Journal of Physics: Conference Series, June 2005

[2] Nathan Bell and Michael Garland "Efficient Sparse Matrix-Vector Multiplication on CUDA" NVIDIA Technical Report NVR-2008-004", December 2008

[3] Samuel Samuel Williams,, Leonid Oliker, Richard Vuduc,, John Shalf,, Katherine Yelick, James Demmel, “Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms “