

毕业论文

中文摘要关键词
AbstractKey words
目录

一、引言

二、研究背景综述

近年来，在工程应用中，求解高阶矩阵的需求日益增长，全矩阵运算脱离了实际的硬件限制，为了满足这一日益增长的需求，同时这些矩阵通常都有着一个特征——非零元远少于零元，稀疏矩阵这门学科便应运而生。在 20 世纪 60 年代研发电子网络的电子工程师们是最早的去利用稀疏性来应用稀疏矩阵进行工程上的计算的。[1] 而在微分方程数值解、线性规划等的有限元分析中，经常出现求解高阶稀疏线性方程组，如利用全矩阵进行存储，则需要 n^2 的空间复杂度和 n^3 的乘法运算时间复杂度，显然，这种程度的运算量是无法被微型计算机，甚至是工作站所接受的。而利用矩阵的稀疏性，可以有效地减小消耗很多无谓的存储空间以及无谓的计算，在很大的程度上降低了时间和空间复杂度，降低了计算对硬件的需求，使计算成为可能。

(1) 稀疏矩阵的定义

稀疏矩阵是非零元远小于矩阵元素总数，且非零元分布没有规律的矩阵。[1]

(2) 稀疏矩阵已有存储方式

在这些年的发展中，出现了很多的存储方法，比如：对角线存储法、对称矩阵的变带宽存储法、坐标存储法、Elipack-Itpack 存储法、CSR 存储法、Shermans 存储法、超矩阵存储法、动态存储方案等 [2]。

如行压缩存储方式 (Compressed Row Storage): CRS 存储可以高效地存取任意一行非零元素，但存取任意一列非零元则需要遍历整个 CRS 存储结构。相应地，与 CRS 存储的稀疏矩阵相关的算法要高效的编程实现，算法的计算顺序必须按行来进行。[3] 下面对应于 CRS 存储：

ROWP	0	2	5	8	9	12						
COLIND	0	1	0	2	4	1	2	3	2	1	2	4
VAL	a	b	c	d	e	f	g	h	i	j	k	l

我们可以发现，ROWP 数组存储的时行非零元的增量。COLIND 则存储的是列索引值，典型的 C 语言结构实现为

```
struct cr_matrix{
    int *Ri; /* col index */
    int *Rp; /* length nrow+1 */
    double *Rx;
    int Rncol;
    int Rnrow;
};
```

(3) 已有的处理稀疏矩阵存储与运算的开源库

在做稀疏矩阵的计算时，通常都是做一系列的基本运算，如：矩阵转置、矩阵向量乘法、矩阵矩阵乘法、数乘等。为了能得到更好的效率，许多研究者致力于寻找对于这些计算最优的存储结构及计算算法，同时提供了许多类库供科学计算使用，如：Portable, Extensible Toolkit for Scientific Computation (PETSc)、Boost、GNU Scientific Library (GSL) 等。

例如，在 Boost-uBLAS 中有着稀疏矩阵的模板 `mapped_matrix<T, F, A>`（元素映射矩阵存储形式）、`compressed_matrix<T, F, IB, IA, TA>`（压缩存储格式）、`coordinat_matrix<T, F, IB, IA, TA>`（坐标存储格式）。分别有着如下示例：[4]

mapped_matrix:

```
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/io.hpp>

int main () {
    using namespace boost::numeric::ublas;
    mapped_matrix<double> m (3, 3, 3 * 3);
    for (unsigned i = 0; i < m.size1 (); ++ i)
        for (unsigned j = 0; j < m.size2 (); ++ j)
            m (i, j) = 3 * i + j;
    std::cout << m << std::endl;
}
```

compressed_matrix:

```
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/io.hpp>

int main () {
    using namespace boost::numeric::ublas;
```

```

        compressed_matrix<double> m (3, 3, 3 * 3);
        for (unsigned i = 0; i < m.size1 (); ++ i)
            for (unsigned j = 0; j < m.size2 (); ++ j)
                m (i, j) = 3 * i + j;
        std::cout << m << std::endl;
    }

    coordinate_matrix:

#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/io.hpp>

int main () {
    using namespace boost::numeric::ublas;
    coordinate_matrix<double> m (3, 3, 3 * 3);
    for (unsigned i = 0; i < m.size1 (); ++ i)
        for (unsigned j = 0; j < m.size2 (); ++ j)
            m (i, j) = 3 * i + j;
    std::cout << m << std::endl;
}

```

三、研究的目标、内容及研究方法

(1) 研究的目标

利用 c++ 在 g++ 编译器 linux 平台下实现稀疏矩阵的行压缩存储方式 (Compressed Row Storage) 存储。可以并行计算稀疏矩阵的矩阵乘法、数乘等运算。

(2) 研究内容

在 linux 平台下, 基于 g++ 编译器, 编写稀疏矩阵存储的 C++ 库, 实现稀疏矩阵的行压缩存储方式 (Compressed Row Storage), 同时实现一些稀疏矩阵的基本运算, 如矩阵乘法、数乘等。为了更好地发挥 CPU 性能, 需要实现多线程并行计算。

(3) 研究方法

实现稀疏矩阵的行压缩存储方式 (Compressed Row Storage), 通过数组存储非零元的增长量、列索引值和非零元值来实现稀疏矩阵的存储。为了实现可变长数组的存储, 利用 C++ 的 STL 中的 vector 来存储数据。

而为了更好地利用 CPU 的性能, 利用 thread 库实现多线程并行计算, 更为有效地利用 CPU 的并行性能。而为了防止多线程并行计算中“脏数据”的出现, 利用 mutex 互斥锁保障进程安全性。

为了实现稀疏矩阵的矩阵乘法、矩阵向量乘法, 可以参考全矩阵矩阵乘法、矩阵向量乘法算法, 利用稀疏矩阵的零元不参与计算的特性, 按照稀疏矩阵存储的结构, 较全矩阵计算省略大量计算, 实现稀疏矩阵计算的优势——更为高效的计算。

四、实验结果

五、结论

参考文献

- [1]Yousef Saad.Iterative Methods for Sparse Linear Systems[M].SECOND EDITION.USA:Society for Industrial and Applied Mathematics,2003 年.68.
- [2] 张永杰、孙秦. 稀疏矩阵存储技术 [J]. 长春理工大学学报,2006 年,03 期:38-41.
- [3] 冯广祥. 大型稀疏矩阵直接求解算法的研究及实现 [8]. 东北大学: 系统工程, 2010.
- [4]Joerg Walter and Mathias Koch.Boost 1.57.0 Library Documentation-uBLAS.
http://www.boost.org/doc/libs/1_57_0/libs/numeric/ublas/doc/index.html, 2011