并行与分布式计算作业

大作业

姓名：郝裕玮

班级：计科1班

学号：18329015

1. 问题描述

实现一个并行的线性求解器，求解线性方程的解

1. 解决方案

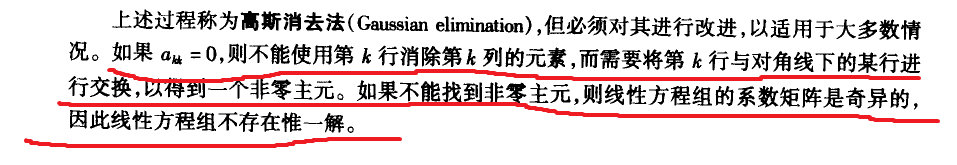
求解线性方程组的通用方法为高斯消元法。具体步骤如下：

（1）将A*x* = b的矩阵写为增广矩阵（A,b）；

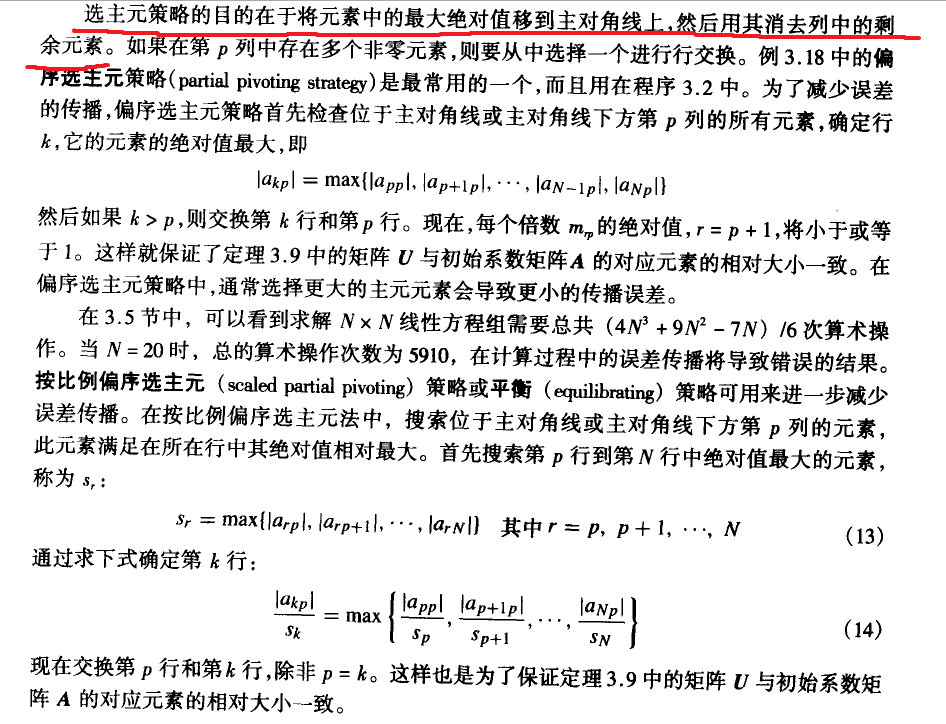
（2）选取每一行的主元并利用该主元消去所在列中的剩余元素，最终可将矩阵转换为上三角矩阵；

（3）利用回代法即可求出方程组的解向量*x*

其中对于（2），注意事项如下：



同时，（2）的选取主元的方法具体为：



由上述方法可知：在将增广矩阵化为上三角时，因为我们每一次只清除一列元素，而每一行的元素需要进行一次数字运算，且行与行之间没有数据依赖，所以该部分可以并行。同时对于回代法同样可以对该部分进行并行化处理。

具体代码实现如下所示（具体分析已包含在注释中）：

#include <iostream>

#include <omp.h>

#include <cstdlib>

#include <ctime>

#include <cmath>

#include <stack>

#include <sys/time.h>

#include <time.h>

using namespace std;

#define GET\_TIME(now) { \

   struct timeval t; \

   gettimeofday(&t, NULL); \

   now = t.tv\_sec + t.tv\_usec/1000000.0; \

}

////最后一行将us转换为s，统一单位

//该结构体用于计算并行计算和串行计算的运行时间

//命令行编译: g++ -fopenmp gauss.cpp

//求解Ax=b

//A矩阵生成

void matGene(double \*A, int size){

    for(int i = 0; i <= size-1; i++){

        for(int j = 0; j <= size-1; j++){

            //将A[i][j]存储到一维数组A[]中

            A[i \* size + j] = rand() % 10;

        }

    }

}

//b向量生成

void vecGene(double \*b, int size){

    for (int i = 0; i <= size-1; i++){

        b[i] = rand() % 5; //A[i]

    }

}

//展示矩阵A

void matShow(double \*A, int size){

    //按行打印矩阵

    for (int i = 0; i <= size-1; i++){

        for (int j = 0; j <= size-1; j++){

            cout << A[i \* size + j] << " ";

        }

        cout << endl;

    }

}

//展示向量b

void vecShow(double \*b, int size){

    for (int i = 0; i < size; i++){

        cout << b[i] << endl;

    }

}

int main() {

    //设置矩阵(n\*n)和向量规模(n\*1)

    int n = 1000;

    //初始化

    double\* A = new double[n \* n + 1];

    double\* b = new double[n + 1];

    int\* serial\_num = new int[n + 1];

    //随机生成矩阵A和向量b

    srand(time(NULL));

    matGene(A, n);

    vecGene(b, n);

    //用于展示初始化矩阵A和向量b

    //cout << "A = " << endl;

    //matShow(A, n);

    //cout << "b = " << endl;

    //vecShow(b, n);

    for(int i = 0; i < n; i++){

        serial\_num[i] = -1;

    }

    stack<int> s1;

    double start,end;

    //记录计算开始时间

    GET\_TIME(start);

    //设置omp并行线程数量

    int num\_threads = 4;

    //将增广矩阵转化为上三角矩阵

    for(int j = 0; j < n; j++){

        double max\_coff = 0;

        int max\_index;

        for(int i = 0; i < n; i++){

            if(serial\_num[i] == -1 && abs(A[i \* n + j]) > abs(max\_coff)){

                max\_coff = A[i \* n + j];

                max\_index = i;

            }

        }

        serial\_num[max\_index] = j;

        s1.push(max\_index);

        #pragma omp parallel for num\_threads(num\_threads)

        for(int i = 0; i < n; i++){

            if(serial\_num[i] == -1){

                double tmp\_coff = A[i \* n + j] / A[max\_index \* n + j];

                A[i \* n + j] = 0;

                for(int k = j + 1; k < n; k++){

                    A[i \* n + k] -= tmp\_coff \* A[max\_index \* n + k];

                }

                b[i] -= tmp\_coff \* b[max\_index];

            }

        }

    }

    //对上三角矩阵进行回代求解

    double\* result = new double[n + 2];

    for(int j = n - 1; j >= 0; j--){

        int this\_index = s1.top();

        s1.pop();

        serial\_num[this\_index] = -1;

        #pragma omp parallel for num\_threads(num\_threads)

        for(int i = 0; i < n; i++){

            if(serial\_num[i] != -1){

                double tmp\_coff = A[i \* n + j] / A[this\_index \* n + j];

                A[i \* n + j] = 0;

                b[i] -= tmp\_coff \* b[this\_index];

            }

        }

        b[this\_index] /= A[this\_index \* n + j];

        A[this\_index \* n + j] = 1;

        result[j] = b[this\_index];

    }

    GET\_TIME(end);

    //可展示解向量x

    //cout << "x = " << endl;

    //vecShow(result, n);

    //输出并行运算时间

    cout << "Time: " << end-start << endl;

    return 0;

}

1. 实验结果

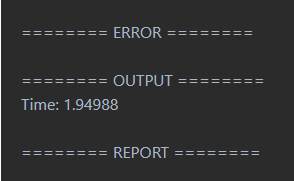
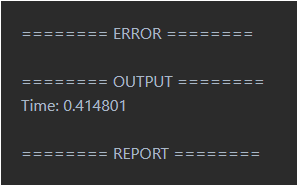
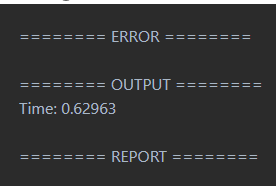
以下结果均在超算习堂上运行得出。

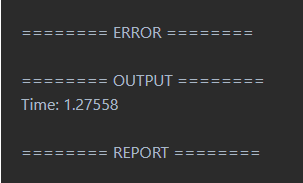
我们只需要将代码中的同样的2行代码

#pragma omp parallel for num\_threads(num\_threads)

注释掉，即可将该程序变为串行程序。

（1）不同线程数量下的运行时间如下图所示：

矩阵规模为1000\*1000，线程数量依次为1（串行）、2、4、8

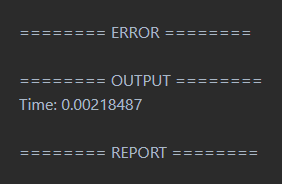
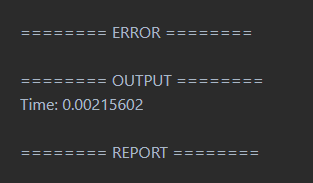


易计算得2,4,8线程相对于串行算法的加速比分别为：1.529，3.097，4.701，即加速比随着线程数量的增加越来越高。

（2）不同矩阵规模下的运行时间如下图所示：

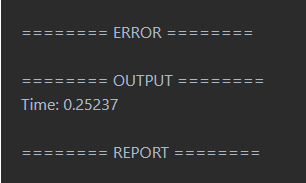
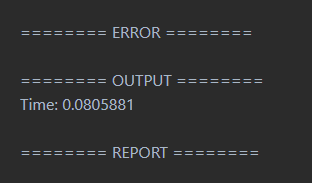
线程数为4，矩阵规模为100\*100，500\*500,1000\*1000

对于100\*100，左图为串行，右图为并行：

所以加速比为：1.013

对于500\*500，左图为串行，右图为并行：

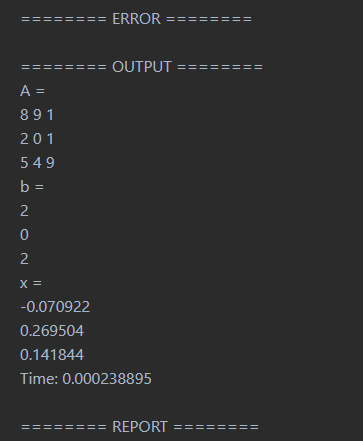
 

所以加速比为：3.132

对于1000\*1000，（1）中已计算过，加速比为3.097。

由上图结果可知，并行加速比随着矩阵规模的增大而增加。

为证明方程组的解向量*x*计算正确，我们取如下方程组进行验证（矩阵规模为3\*3）：

易证计算结果正确（右图网址为：<http://www.ab126.com/shuxue/2693.html>）

四、遇到的问题及解决方法

本次实验由于之前有在数值计算上学习过高斯消元法的相关知识，所以只需对着书本即可实现串行算法而在此基础上的OpenMP并行算法也并不难，只需要在两个for循环部分加上OpenMP的并行语句即可。