# 实验报告五

写一个多线程稀疏矩阵乘法运算函数。此运算函数使用P（常量）个线程同时计算。稀疏矩阵中很多位置的值为零，在进行浮点数运算时，如有操作数为零，运算速度更快。为了尽可能的均匀分配计算任务，不能固定分配计算任务。

输入参数：float[][]A, float[][]B，float[][]C A的大小为 Ra×Ca，B的大小为Rb×Cb , 其中Ca=Rb。 C的大小为Ra×Cb，结果放在C中

假设Ra，Ca，Rb，Cb都是预先定义好的常量。

要求：动态调整每个线程处理的数据量，使得每个线程基本同时结束。请说明算法的调度策略。进行调度时，如果涉及到数据访问冲突，请正确的加锁。

提示：每个线程可以在计算前申请需要计算的目标矩阵中的位置。计算完毕后写入目标向量。

1. 题目分析
2. 代码实现

package test05;

import java.util.concurrent.atomic.AtomicInteger;

public class Matrix\_Thread implements Runnable {

private static final int Ra = 2;

private static final int Ca = 5;

private static final int Rb = 5;

private static final int Cb = 2;

private static final int P = 4;

private static float[][] A = new float[Ra][Ca];

private static float[][] B = new float[Rb][Cb];

private static float[][] C = new float[Ra][Cb];

// 原子变量，用于记录当前已经计算的行数

private static AtomicInteger currentRow = new AtomicInteger(0);

public static void initMatrix() {

for (int i = 0; i < Ra; i++) {

for (int j = 0; j < Ca; j++) {

A[i][j] = (float) Math.random();

}

}

for (int i = 0; i < Rb; i++) {

for (int j = 0; j < Cb; j++) {

B[i][j] = (float) Math.random();

}

}

}

// 计算C矩阵中row行中每一列的元素

public void computeRow(int row) {

for (int col = 0; col < Cb; col++) {

float sum = 0;

for (int k = 0; k < Ca; k++) {

sum += A[row][k] \* B[k][col];

}

C[row][col] = sum;

}

}

@Override

public void run() {

while (true) {

int row = currentRow.getAndIncrement();

if (row >= Ra) {

break;

}

computeRow(row);

}

}

public static void main(String[] args) throws InterruptedException {

initMatrix();

Thread[] threads = new Thread[P];

for (int i = 0; i < P; i++) {

threads[i] = new Thread(new Matrix\_Thread());

threads[i].start();

}

for (int i = 0; i < P; i++) {

//先将当前执行线程挂起，待其他线程结束后继续执行当前线程

//保证打印结果的准确性

threads[i].join();

}

for (int i = 0; i < Ra; i++) {

for (int j = 0; j < Cb; j++) {

System.out.print(C[i][j] + " ");

}

System.out.println();

}

}

}

1. 运行结果

