# JAVA编程进阶上机报告

****

**学 院 智能与计算学部**

**专 业\_软件工程\_\_\_\_\_\_**

**姓 名\_王传安\_\_\_\_\_\_\_\_**

**学 号\_3018216301\_\_\_\_**

**年 级\_2018级 \_\_\_\_\_\_\_**

**班 级\_软工六班\_\_\_\_\_\_**

1. **实验要求**

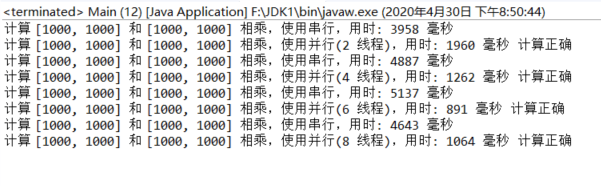
编写矩阵随机生成类 MatrixGenerator 类，随机生成任意大小的矩阵，矩阵单元使用 double 存储。

使用串行方式实现矩阵乘法。

使用多线程方式实现矩阵乘法。

比较串行和并行两种方式使用的时间，利用第三次使用中使用过的 jvm状态查看命令，分析产生时间差异的原因是什么。

**二、实验结果**



在使用2线程的时候，用时大约是串行的一半，但是随着线程数量的增加，用时减少的非常少，并且在使用8线程时，用时比6线程还要长，这是因为虽然多线程加快了计算，但是线程本身创建的时间也造成了一部分开销。

1. **实验代码**

部分代码如下(完整代码见压缩包内)

**public** **class** MatrixThread **implements** Runnable{

**private** **int** start;

**private** **double**[][] A;

**private** **double**[][] B;

**private** **double**[][] C;

**public** MatrixThread(**int** start,**double**[][] A,**double**[][] B,**double**[][] C){

**this**.start=start;

**this**.A = A;

**this**.B = B;

**this**.C = C;

}

@Override

**public** **void** run() {

**int** i,j,k;

**for**(i=start; i<Driver.*M*; i += Driver.*NUM\_THREADS*){

**for**(j=0;j<Driver.*N*;j++){

**for**( k=0; k< Driver.*K*;k++) {

C[i][j]+=A[i][k]\*B[k][j];

}

}

}

}

}

**public** **class** Driver {

**public** **static** **int** *M*;

**public** **static** **int** *K*;

**public** **static** **int** *N*;

**public** **static** **int** *NUM\_THREADS*;

**private** **static** **double**[][] *A*;

**private** **static** **double**[][] *B*;

**private** **static** **double**[][] *C1*;

**private** **static** **double**[][] *C2*;

**public** Driver(**int** M, **int** K, **int** N, **int** NUM\_THREADS, **double**[][] A, **double**[][] B){

**this**.*M* = M;

**this**.*K* = K;

**this**.*N* = N;

**this**.*NUM\_THREADS* = NUM\_THREADS;

**this**.*A* = A;

**this**.*B* = B;

*C1* = **new** **double** [M][N];

*C2* = **new** **double** [M][N];

**for**(**int** i=0;i<M;i++) {

**for**(**int** j=0;j<N;j++) {

*C1*[i][j]=0;

*C2*[i][j]=0;

}

}

}

**public** **static** **void** singleThread(){

**for**(**int** i=0;i<*M*;i++){

**for**(**int** j=0;j<*N*;j++){

**for**(**int** k=0;k<*K*;k++) {

*C1*[i][j] += *A*[i][k] \* *B*[k][j];

}

}

}

}

**public** **void** Serial() {

**long** time1 = System.*currentTimeMillis*();

*singleThread*();

**long** time2 = System.*currentTimeMillis*();

System.***out***.println("计算 [" + *M* + ", " + *N* + "] 和 [" + *N* + ", " + *K* + "] 相乘，使用串行，用时: " + (time2 - time1) + " 毫秒");

}

**public** **void** Parallel() {

Thread[] workers=**new** Thread[*NUM\_THREADS*];

**for**(**int** i=0;i<*NUM\_THREADS*;i++) {

workers[i]=**new** Thread(**new** MatrixThread(i,*A*,*B*,*C2*));

}

**long** time1= System.*currentTimeMillis*();

**for**(**int** i=0; i<*NUM\_THREADS*; i++){

workers[i].start();

}

**for**(**int** i=0;i<*NUM\_THREADS*;i++){

**try**{

workers[i].join();

}**catch**(InterruptedException e){

e.printStackTrace();

}

}

**long** time2 = System.*currentTimeMillis*();

System.***out***.print("计算 [" + *M* + ", " + *N* + "] 和 [" + *N* + ", " + *K* + "] 相乘，使用并行(" + *NUM\_THREADS* + " 线程)，用时: " + (time2 - time1) + " 毫秒 ");

judge();

System.***out***.print("\r\n");

}

**public** **void** judge() {

**boolean** isRight = **true**;

**for**(**int** i=0; i<*M*; i++) {

**for**(**int** j=0; j<*N*; j++) {

**if**(Math.*abs*(*C1*[i][j] - *C2*[i][j]) > 0.5) {

isRight = **false**;

**break**;

}

}

}

**try** {

**assert** isRight : "计算错误";

System.***out***.print("计算正确");

}**catch** (AssertionError err) {

System.***out***.println(err.getMessage());

}

}

}