Оптимизация

1. Без оптимизации

Ф

private static double[][] multiply(double[][] m1, double[][] m2) {

double[][] result = new double[m1.length][m2[0].length];

for (int i = 0; i < m1[0].length; i++) {

for (int j = 0; j < m1.length; j++) {

double summand = 0.0;

for (int k = 0; k < m2.length; k++) {

summand += m1[i][k] \* m2[k][j];

}

result[i][j] = summand;

}

}

return result;

}

---------------------------------------

Apache library test:

Initialisation:

128

Calculation:

8458

---------------------------------------

---------------------------------------

HandMade library test:

Initialisation:

77

Calculation:

9759

---------------------------------------

1. Final размерность

Ф

private static double[][] multiply(double[][] m1, double[][] m2) {

double[][] result = new double[m1.length][m2[0].length];

final int aColumns = m1.length;

final int aRows = m1[0].length;

final int bColumns = m2.length;

final int bRows = m2[0].length;

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

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

double summand = 0.0;

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

summand += m1[i][k] \* m2[k][j];

}

result[i][j] = summand;

}

}

return result;

}

---------------------------------------

HandMade library test:

Initialisation:

93

Calculation:

8925

---------------------------------------

1. Использование транспонированной матрицы для лучшего использования Кэша

Ф

private static double[][] multiply(double[][] m1, double[][] m2) {

double[][] result = new double[m1.length][m2[0].length];

final int aColumns = m1.length;

final int aRows = m1[0].length;

final int bColumns = m2.length;

final int bRows = m2[0].length;

double m2T[][] = new double[bColumns][bRows];

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

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

m2T[j][i] = m2[i][j];

}

}

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

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

double summand = 0.0;

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

summand += m1[i][k] \* m2T[j][k];

}

result[i][j] = summand;

}

}

return result;

}

---------------------------------------

HandMade library test:

Initialisation:

89

Calculation:

1480

---------------------------------------

1. Попытка объединить циклы

Ф

private static double[][] multiply(double[][] m1, double[][] m2) {

double[][] result = new double[m1.length][m2[0].length];

final int aColumns = m1.length;

final int aRows = m1[0].length;

final int bColumns = m2.length;

final int bRows = m2[0].length;

double thatColumn[] = new double[bRows];

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

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

thatColumn[k] = m2[k][j];

}

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

double thisRow[] = m1[i];

double summand = 0;

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

summand += thisRow[k] \* thatColumn[k];

}

result[i][j] = summand;

}

}

return result;

}

---------------------------------------

HandMade library test:

Initialisation:

89

Calculation:

1714

---------------------------------------

1. Пробуем отлавливать эксепшен вместо проверки на выход

Ф

private static double[][] multiply(double[][] m1, double[][] m2) {

double[][] result = new double[m1.length][m2[0].length];

final int aColumns = m1.length;

final int aRows = m1[0].length;

final int bColumns = m2.length;

final int bRows = m2[0].length;

double m2T[][] = new double[bColumns][bRows];

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

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

m2T[j][i] = m2[i][j];

}

}

try{

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

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

double summand = 0.0;

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

summand += m1[i][k] \* m2T[j][k];

}

result[i][j] = summand;

}

}

}catch (IndexOutOfBoundsException ignored) { }

return result;

}

---------------------------------------

HandMade library test:

Initialisation:

90

Calculation:

1489

---------------------------------------

1. ф