МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**по дисциплине  
 «Параллельное и низкоуровневое программирование»**

Выполнил студент группы 25/2                                       А.А. Козин

Направление подготовки  02.03.03  Математическое обеспечение и администрирование информационных систем

Курс    2

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2022 г.

**Задание**: средствами OMP распараллелить процесс нахождения обратной матрицы.  
**Решение**: алгоритм нахождения обратной матрицы реализован через транспонирование матрицы, затем нормированием по столбцам и строкам. Процедура повторяется до заданной точности. Чтобы алгоритм работал, нужно сначала ввести размер матрицы, далее предоставляется выбор: “1” – ввод вручную или “2” – сгенерировать случайно. Когда все нужные данные введены начинается работа алгоритма. В консоль выводиться начальная матрица и обратная через функцию display. Для замера времени используются переменные t0 и t1, результат затраченного времени выводиться в консоль.

**Текст программы:**

#include <iostream>

#include <ctime>

#include <omp.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

double\*\* copy(double\*\* array, int n) {

double\*\* newArray = new double\* [n];

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

newArray[i] = new double[n];

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

newArray[i][j] = array[i][j];

}

}

return newArray;

}

double\*\* matrixMultiple(double\*\* matrixA, double\*\* matrixB, int n) {

double\*\* res = new double\* [n];

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

res[i] = new double[n];

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

res[i][j] = 0;

}

}

#pragma omp parallel for shared (res,matrixA,matrixB)

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

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

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

res[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

return res;

}

void scalarMultiple(double\*\* matrix, int n, double x) {

#pragma omp parallel for shared (x,matrix)

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

matrix[i][j] \*= x;

}

void suma(double\*\* matrixA, double\*\* matrixB, int n) {

#pragma omp parallel for shared (matrixA,matrixB)

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

matrixA[i][j] += matrixB[i][j];

}

void deleteArray(double\*\* array, int n) {

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

delete[] array[i];

}

delete[] array;

}

double determinant(double\*\* matrix, int n) {

double\*\* matrixB = copy(matrix, n);

for (int temp = 0; temp < n - 1; temp++) {

for (int i = temp + 1; i < n; i++) {

double coeff = -matrixB[i][temp] / matrixB[temp][temp];

for (int j = temp; j < n; j++) {

matrixB[i][j] += matrixB[temp][j] \* coeff;

}

}

}

double Det = 1;

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

Det \*= matrixB[i][i];

}

deleteArray(matrixB, n);

return Det;

}

void display(double\*\* matrix, int n) {

cout << "matrix:" << endl;

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

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

cout << matrix[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

void main() {

cout << "Enter number of equations: ";

int n; cin >> n;

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

cout << "Do you want to enter values manually or randomly ?" << endl << "1 - manually, 2 - random" << endl;

int input; cin >> input; cout << endl;

switch (input) {

case 1:

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

A[i] = new double[n];

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

cin >> A[i][j];

}

}

cout << endl;

break;

case 2:

srand(time(0));

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

A[i] = new double[n];

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

A[i][j] = rand() % 10;

}

}

cout << endl;

break;

default:

cout << "Error, bad input, quitting." << endl;

break;

}

display(A, n);

auto t0 = high\_resolution\_clock::now();

double num0 = 0, num1 = 0;

double\*\* copyA = copy(A, n);

for (size\_t i = 0; i < n; i++) {

double sumj = 0, sumi = 0;

for (size\_t j = 0; j < n; j++) {

sumi += fabs(copyA[i][j]);

sumj += fabs(copyA[j][i]);

}

num0 = max(sumj, num0);

num1 = max(sumi, num1);

}

for (size\_t i = 0; i < n - 1; i++) {

for (size\_t j = i + 1; j < n; j++) {

swap(copyA[j][i], copyA[i][j]);

}

}

double\*\* Ez = new double\* [n];

scalarMultiple(copyA, n, (1 / (num0 \* num1)));

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

Ez[i] = new double[n];

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

if (i == j)

Ez[i][j] = 2;

else

Ez[i][j] = 0;

}

}

double e = 0.001;

double\*\* inverse = copy(copyA, n);

if (determinant(A, n) != 0) {

while (fabs(determinant(matrixMultiple(A, inverse, n), n) - 1) >= e) {

double\*\* pred = copy(inverse, n);

inverse = matrixMultiple(A, pred, n);

scalarMultiple(inverse, n, -1);

suma(inverse, Ez, n);

inverse = matrixMultiple(pred, inverse, n);

deleteArray(pred, n);

}

cout << "Inverse ";

display(inverse, n);

}

else

cout << "Inverse matrix does not exist. ";

auto t1 = high\_resolution\_clock::now();

deleteArray(A, n);

deleteArray(Ez, n);

cout << "duration\_cast: " << duration\_cast<milliseconds>(t1 - t0).count() << " milliseconds" << endl;

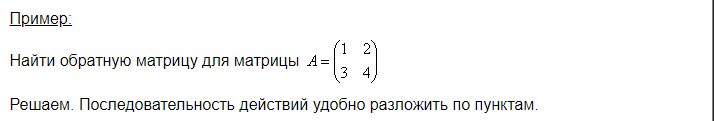
}

**Программа выдает в консоли результаты в следующем виде:**

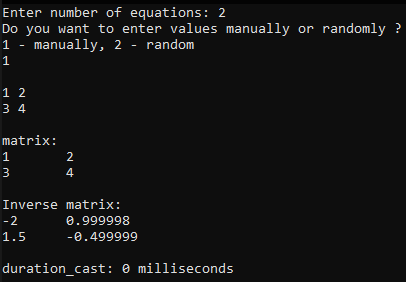
Для проверки верности решения возьму примеры из [mathprofi](http://mathprofi.ru/kak_naiti_obratnuyu_matricu.html).

Решение примеров можно посмотреть, перейдя по ссылке выше.

Пример 1:

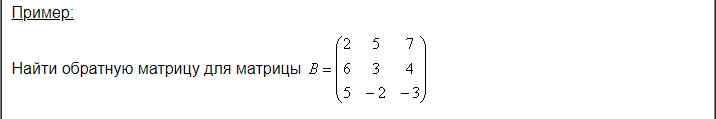


Ввожу значения:

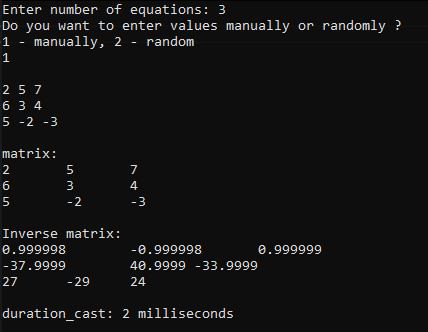


Ответ приблизительно верный: .

Пример 2:



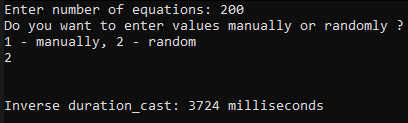
Ввожу значения:



Ответ так же приблизительно правильный:.

Теперь посмотрим будет ли эффективнее алгоритм с OMP. Для этого зададим матрицы большого размера рандомно и рекомендую закомментировать строки 118 и 162 (с вызовом функции display, чтобы не видеть огромные матрицы).

Без OMP:



С OMP:

