Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 Информатика и вычислительная техника

Дисциплина: «Технологии блокчейн и распределенные информационные системы»

**ОТЧЕТ**

по лабораторной работе №7

Выполнил: студент группы АСУ-19-1б

Нерослов Е. А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Щапов В. А. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2023

ЗАДАНИЕ

Реализовать решение СЛАУ различными методами с помощью IntelTBB.

КОД

main.cpp

#include <iostream>

#include <tbb/tbb.h>

#include "MatrixMethod.h"

#include "GaussMethod.h"

using namespace tbb;

using namespace std;

int main()

{

system("chcp 1251");

system("cls");

//srand(time(NULL));

int equations\_amount;

cout << "Введите количество уравнений: ";

cin >> equations\_amount;

cout << "Матричный метод:\n";

MatrixMethod matrix\_method(equations\_amount);

matrix\_method.run\_matrix\_method();

cout << "Метод Гаусса:\n";

GaussMethod gauss\_method(equations\_amount);

gauss\_method.run\_gauss\_method();

}

MatrixMethod.h

#include <iostream>

#include <time.h>

#include <vector>

#include <tbb/tbb.h>

using namespace std;

using namespace tbb;

#pragma once

class MatrixMethod

{

private:

int equations\_amount;

public:

MatrixMethod(int n)

{

equations\_amount = n;

}

private:

double random(const int min, const int max)

{

if (min == max)

return min;

return min + rand() % (max - min);

}

vector<double> generate\_random\_double\_array(int size)

{

vector<double> random\_array;

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

{

double random\_number = random(0, 100);

random\_array.push\_back(random\_number);

}

return random\_array;

}

vector<vector<double>> generate\_random\_double\_matrix(int size)

{

vector<vector<double>> matrix;

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

{

vector<double> row = generate\_random\_double\_array(size);

matrix.push\_back(row);

}

return matrix;

}

void print\_matrix(vector<vector<double>> matrix)

{

for (size\_t i = 0; i < matrix.size(); i++)

{

cout << endl;

for (size\_t j = 0; j < matrix.size(); j++)

{

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

}

}

cout << endl

<< endl;

}

void print\_array(vector<double> array)

{

for (size\_t i = 0; i < array.size(); i++)

{

cout << array[i] << endl;

}

}

// Функция вычисления обратной матрицы

vector<vector<double>> search\_reverse\_matrix(vector<vector<double>> matrix, int n) {

// Объявляем матрицу для хранения результата

vector<vector<double>> result(n, vector<double>(n));

// Обратная матрица равна единичной матрице

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

result[i][i] = 1;

// Находим обратную матрицу

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

// Находим детерминант матрицы

double det = matrix[i][i];

// Инвертируем детерминант

det = 1.0 / det;

// Находим обратную матрицу

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

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

result[i][j] \*= det;

}

// Делим строку на детерминант

parallel\_for(tbb::blocked\_range<int>(0, n),

[&](const tbb::blocked\_range<int>& r) {

for (int j = r.begin(); j != r.end(); ++j) {

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

result[i][j] \*= det;

}

});

// Вычитаем строку из остальных строк

parallel\_for(tbb::blocked\_range<int>(0, n),

[&](const tbb::blocked\_range<int>& r) {

for (int k = r.begin(); k != r.end(); ++k) {

if (k != i) {

double c = matrix[k][i];

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

matrix[k][j] -= matrix[i][j] \* c;

result[k][j] -= result[i][j] \* c;

}

}

}

});

}

return matrix;

}

public:

void run\_matrix\_method()

{

vector<vector<double>> a = generate\_random\_double\_matrix(equations\_amount);

vector<double> b = generate\_random\_double\_array(equations\_amount);

vector<double> x(equations\_amount);

// Нахождение обратной матрицы

vector<vector<double>> E(equations\_amount, vector<double>(equations\_amount));

for (int i = 0; i < equations\_amount; ++i) {

E[i][i] = 1;

}

tick\_count beg = tick\_count::now();

E = search\_reverse\_matrix(a, equations\_amount);

// Нахождение ответа

parallel\_for(blocked\_range<size\_t>(0, equations\_amount),

[&](const blocked\_range<size\_t>& r) {

for (size\_t i = r.begin(); i < r.end(); ++i) {

double sum = 0;

for (int j = 0; j < equations\_amount; ++j) {

sum += E[i][j] \* b[j];

}

x[i] = sum;

}

});

tick\_count end = tick\_count::now();

printf("Время, затраченное на вычисление: %f с.\n", (end - beg).seconds());

}

};

GaussMethod.h

#include <iostream>

#include <vector>

#include <time.h>

#include <tbb/tbb.h>

using namespace std;

using namespace tbb;

#pragma once

class GaussMethod

{

private:

int equations\_amount;

public:

GaussMethod(int n)

{

equations\_amount = n;

}

private:

double random(const int min, const int max)

{

if (min == max)

return min;

return min + rand() % (max - min);

}

vector<double> generate\_random\_double\_array(int size)

{

vector<double> random\_array;

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

{

double random\_number = random(0, 100);

random\_array.push\_back(random\_number);

}

return random\_array;

}

vector<vector<double>> generate\_random\_double\_matrix(int size)

{

vector<vector<double>> matrix;

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

{

vector<double> row = generate\_random\_double\_array(size);

matrix.push\_back(row);

}

return matrix;

}

vector<vector<double>> generate\_identity(int size)

{

vector<vector<double>> I;

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

{

vector<double> row;

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

{

if (i == j)

{

row.push\_back(1);

continue;

}

row.push\_back(0);

}

I.push\_back(row);

}

return I;

}

void print\_matrix(vector<vector<double>> matrix)

{

for (size\_t i = 0; i < matrix.size(); i++)

{

cout << endl;

for (size\_t j = 0; j < matrix.size(); j++)

{

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

}

}

cout << endl

<< endl;

}

public:

void run\_gauss\_method()

{

// Матрица

double\*\* A;

A = new double\* [equations\_amount];

for (int i = 0; i < equations\_amount; i++)

A[i] = new double[equations\_amount];

// Вектор правой части

double\* B = new double[equations\_amount];

// Заполняем матрицу коэффициентов и B

for (int i = 0; i < equations\_amount; i++)

{

for (int j = 0; j < equations\_amount; j++)

A[i][j] = random(0, 100);

B[i] = random(0, 100);

}

tick\_count beg = tick\_count::now();

// Создаем параллельный контекст

parallel\_for(0, equations\_amount, 1, [&](int i) {

// Находим максимальный элемент в столбце

double max = 0.0;

for (int j = 0; j < equations\_amount; j++) {

if (A[i][j] > max)

max = A[i][j];

}

// Делим строку на максимальный элемент

for (int j = 0; j < equations\_amount; j++) {

A[i][j] /= max;

}

B[i] /= max;

});

// Вычитаем строку из всех последующих строк

parallel\_for(0, equations\_amount - 1, 1, [&](int i) {

for (int j = i + 1; j < equations\_amount; j++) {

double m = A[j][i] / A[i][i];

for (int k = 0; k < equations\_amount; k++) {

A[j][k] -= m \* A[i][k];

}

B[j] -= m \* B[i];

}

});

// Находим решение

double\* X = new double[equations\_amount];

for (int i = equations\_amount - 1; i >= 0; i--) {

X[i] = B[i];

for (int j = i + 1; j < equations\_amount; j++) {

X[i] -= A[i][j] \* X[j];

}

X[i] /= A[i][i];

}

tick\_count end = tick\_count::now();

printf("Время, затраченное на вычисление: %f с.\n", (end - beg).seconds());

}

};

ТЕСТЫ

|  |  |  |
| --- | --- | --- |
| Кол. ур. | Матричный метод, с | Метод Гаусса, с |
| 100 | 0,042895 | 0,002478 |
| 200 | 0,14 | 0,006574 |
| 300 | 0,435043 | 0,016094 |
| 400 | 0,87899 | 0,036738 |
| 500 | 1,575131 | 0,068241 |
| 600 | 2,808085 | 0,117089 |
| 700 | 4,546692 | 0,274684 |
| 800 | 8,687344 | 0,318462 |
| 900 | 10,519626 | 0,411919 |
| 1000 | 15,985155 | 0,775111 |

|  |  |  |
| --- | --- | --- |
| Кол. ур. | Матричный метод, Размер/с | Метод Гаусса, Размер/с |
| 100 | 2331,274041 | 40355,1251 |
| 200 | 1428,571429 | 30422,878 |
| 300 | 689,5870063 | 18640,48714 |
| 400 | 455,0677482 | 10887,90898 |
| 500 | 317,433915 | 7326,97352 |
| 600 | 213,6687458 | 5124,307151 |
| 700 | 153,9580865 | 2548,382869 |
| 800 | 92,08798454 | 2512,073654 |
| 900 | 85,55437237 | 2184,895574 |
| 1000 | 62,55804213 | 1290,1378 |