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 №4

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 OpenMP.

КОД

main.cpp

#include <iostream>

#include "MatrixMethod.h"

#include "GaussMethod.h"

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 <omp.h>

#include <vector>

using namespace std;

#pragma once

class MatrixMethod

{

private:

int equations\_amount;

public:

MatrixMethod(int n)

{

equations\_amount = n;

}

private:

bool search\_reverse\_matrix(vector <vector<double>>& matrix)

{

int size = matrix.size();

vector <vector<double>> E(size, vector<double>(size));

//Заполнение единичной матрицы

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

{

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

{

if (i == j) E[i][j] = 1.0;

else E[i][j] = 0.0;

}

}

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

{

if (abs(matrix[k][k]) < 1e-8)

{

bool changed = false;

for (int i = k + 1; i < size; i++)

{

if (abs(matrix[i][k]) > 1e-8)

{

swap(matrix[k], matrix[i]);

swap(E[k], E[i]);

changed = true;

break;

}

}

if (!changed)

return false;

}

double div = matrix[k][k];

#pragma omp parallel

{

#pragma omp for

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

{

matrix[k][j] /= div;

E[k][j] /= div;

}

}

#pragma omp parallel

{

#pragma omp for

for (int i = k + 1; i < size; i++)

{

double multi = matrix[i][k];

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

{

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

E[i][j] -= multi \* E[k][j];

}

}

}

}

//Формирование единичной матрицы из исходной

//и обратной из единичной

for (int k = size - 1; k > 0; k--)

{

#pragma omp parallel

{

#pragma omp for

for (int i = k - 1; i > -1; i--)

{

double multi = matrix[i][k];

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

{

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

E[i][j] -= multi \* E[k][j];

}

}

}

}

matrix = E;

return true;

}

double random(const int min, const int max)

{

if (min == max)

return min;

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

}

public:

void run\_matrix\_method()

{

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

vector<double> B(equations\_amount);

// Заполняем матрицу коэффициентов и B

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

{

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

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

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

}

double seconds = clock();

// Вычисление обратной матрицы

if (!search\_reverse\_matrix(matrix))

{

cout << "\nНевозможно найти обратную матрицу!\n";

exit(1);

}

// Матрица-столбец неизвестных X и вычисление окончательного результата

vector<double> X(equations\_amount);

#pragma omp parallel

{

#pragma omp for

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

{

X[i] = 0;

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

X[i] += matrix[i][j] \* B[j];

}

}

seconds = (clock() - seconds) / 1000;

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

}

};

GaussMethod.h

#include <iostream>

#include <vector>

#include <time.h>

#include <omp.h>

using namespace std;

#pragma once

class GaussMethod

{

private:

int equations\_amount;

public:

GaussMethod(int n)

{

equations\_amount = n;

}

private:

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

{

vector<double> random\_array;

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

{

double random\_number = (rand() % 99) + 10;

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;

}

vector<vector<double>> generate\_inverse\_parallel(vector<vector<double>> input\_matrix)

{

int size = input\_matrix.size();

vector<vector<double>> I = generate\_identity(size);

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

{

if (input\_matrix[i][i] == 0)

{

for (int j = i + 1; j < size; j++)

{

if (input\_matrix[j][i] != 0.0)

{

swap(input\_matrix[i], input\_matrix[j]);

break;

}

if (j == size - 1)

{

cout << "Inverse does not exist for this matrix";

exit(0);

}

}

}

double scale = input\_matrix[i][i];

#pragma omp parallel for

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

{

input\_matrix[i][col] = input\_matrix[i][col] / scale;

I[i][col] = I[i][col] / scale;

}

if (i < size - 1)

{

#pragma omp parallel for

for (int row = i + 1; row < size; row++)

{

double factor = input\_matrix[row][i];

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

{

input\_matrix[row][col] -= factor \* input\_matrix[i][col];

I[row][col] -= factor \* I[i][col];

}

}

}

}

for (int zeroing\_col = size - 1; zeroing\_col >= 1; zeroing\_col--)

{

#pragma omp parallel for

for (int row = zeroing\_col - 1; row >= 0; row--)

{

double factor = input\_matrix[row][zeroing\_col];

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

{

input\_matrix[row][col] -= factor \* input\_matrix[zeroing\_col][col];

I[row][col] -= factor \* I[zeroing\_col][col];

}

}

}

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()

{

double dtime;

vector<vector<double>> matrix;

vector<vector<double>> matrix\_parallel;

matrix = generate\_random\_double\_matrix(equations\_amount);

double seconds = clock();

matrix\_parallel = generate\_inverse\_parallel(matrix);

seconds = (clock() - seconds) / 1000;

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

}

};

ТЕСТЫ

|  |  |  |
| --- | --- | --- |
| Кол. ур. | Матричный метод, с | Метод Гаусса, с |
| 100 | 0,035 | 0,039 |
| 200 | 0,12 | 0,106 |
| 300 | 0,42 | 0,445 |
| 400 | 0,778 | 0,787 |
| 500 | 1,419 | 1,486 |
| 600 | 2,581 | 2,632 |
| 700 | 4,143 | 5,264 |
| 800 | 7,021 | 9,536 |
| 900 | 11,194 | 13,792 |
| 1000 | 16,068 | 18,385 |

|  |  |  |
| --- | --- | --- |
| Кол. ур. | Матричный метод, Размер/с | Метод Гаусса, Размер/с |
| 100 | 2857,142857 | 2564,102564 |
| 200 | 1666,666667 | 1886,792453 |
| 300 | 714,2857143 | 674,1573034 |
| 400 | 514,1388175 | 508,2592122 |
| 500 | 352,3608175 | 336,473755 |
| 600 | 232,4680356 | 227,9635258 |
| 700 | 168,959691 | 132,9787234 |
| 800 | 113,9438826 | 83,89261745 |
| 900 | 80,4002144 | 65,25522042 |
| 1000 | 62,23549913 | 54,39216753 |