Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

**(ПНИПУ)**

Факультет: Электротехнический

Кафедра: Информационные технологии и автоматизированные системы.

# Направление подготовки: Разработка программно-информационных систем.

**Лабораторная работа № 2**

Технология OpenMP. Решение СЛАУ методом Гаусса.

Выполнил студент гр. РИС-18-1б

Гилязов Р.А.

(Фамилия, имя, отчество)

###### Проверил:

Старший преподаватель кафедры ИТАС, Щапов В.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О. руководителя от кафедры)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

**Пермь, 2020**

**Постановка задачи:**

1. Написать программу, которая будет решать СЛАУ методом Гаусса.
2. Используя технологию OpenMP, распараллелить решение.

**Решение:**

#include <iostream>

#include <random>

#include <chrono>

using namespace std;

// Необязательная функция!!!

// отключение асинхронного i/o в консоль

// развязывание ручного ввода, т.к. он не используется

const int ZERO = []() {

ios\_base::sync\_with\_stdio(false);

/\*returned before cin tie\*/ cin.tie(nullptr);

return 0;

}();

double\*\* get\_triangle\_matrix(double\*\* matr, long size)

{

for (long i = 0; i < size - 1; i++)

{

double curr = matr[i][i];

#pragma omp parallel for

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

{

double del = matr[k][i] / curr;

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

{

matr[k][j] -= matr[i][j] \* del;

}

}

}

return matr;

}

void print\_m(double\*\* matr, long size)

{

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

{

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

{

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

}

cout << " | " << matr[i][size] << endl;

}

}

void print\_r(double\* res, long size)

{

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

{

cout << "x" <<i+1<<" = "<<res[i] << endl;

}

}

double\* get\_result\_slau(double\*\* nmatr, long size)

{

double\* res = new double[size];

#pragma omp parallel for

for (long i = 0; i < size - 1; i++)

{

res[i] = 0;

}

res[size - 1] = nmatr[size - 1][size] / nmatr[size - 1][size - 1];

for (long i = size - 2; i >= 0; i--)

{

double crisp = 0;

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

{

crisp += nmatr[i][j] \* res[j];

}

res[i] = (nmatr[i][size] - crisp) / nmatr[i][i];

}

return res;

}

bool is\_it\_has\_zero\_str(double\*\* nmatr, long size)

{

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

{

bool flag = false;

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

{

if (nmatr[i][j] != 0) {

flag = true; break;

}

}

if (!flag) return false;

}

return true;

}

bool hasLinNonDependence(double\*\* matr, long pos, long size)

{

if (pos == 0) return false;

double coef = matr[0][pos - 1] / matr[0][pos];

bool f = false;

for (int i = pos; i > 0; --i)

{

for (int j = 0; j < size + 1; i++)

{

if (coef != matr[i - 1][j] / matr[i][j])

break;

else if (j == size) return true;

}

}

return false;

}

bool generate(double\*\* matr, long size)

{

default\_random\_engine generator;

tr1::normal\_distribution<double> randomize(-10, 100);

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

{

for (int j = 0; j < size + 1; ++j)

{

matr[i][j] = randomize(generator);

}

if (hasLinNonDependence(matr, i, size)) {

return false;

}

}

return true;

}

int main()

{

long size\_slau = 2000; //количество переменных и уравнений слау

double\*\* matrix = new double\* [size\_slau];

for (long i = 0; i < size\_slau; ++i)

{

matrix[i] = new double[size\_slau + 1];

}

while (!generate(matrix, size\_slau));

/\*tests 1.\*/

// ans == {1;-1;1}

// size\_slau = 3;

//double a1[3][3] = { {2, -1, -6},{7, -4, 2},{1, -2, -4} };

//double b1[3] = {-3, 13, -1};

//for (long i = 0; i < size\_slau; ++i)

//{

// for (long j = 0; j < size\_slau; ++j)

// {

// matrix[i][j] = a1[i][j];

// }

// matrix[i][size\_slau] = b1[i];

//}

/\*tests 2.\*/

// ans == {1;2;2,0}

//size\_slau = 4;

//double a2[4][4] = { {2, 5, 4, 1},{1, 3, 2, 1},{2, 10, 9, 7},{3, 8, 9, 2} };

//double b2[4] = {20,11,40,37};

//for (long i = 0; i < size\_slau; ++i)

//{

// for (long j = 0; j < size\_slau; ++j)

// {

// matrix[i][j] = a2[i][j];

// }

// matrix[i][size\_slau] = b2[i];

//}

/\*tests 3.\*/

// ans == {4, 0, -1}

//size\_slau = 3;

//double a2[3][3] = { {1, 2, 3},{2, -1, 2},{1, 1, 5} };

//double b2[3] = {1,6,-1};

//for (long i = 0; i < size\_slau; ++i)

//{

// for (long j = 0; j < size\_slau; ++j)

// {

// matrix[i][j] = a2[i][j];

// }

// matrix[i][size\_slau] = b2[i];

//}

auto beg = chrono::high\_resolution\_clock::now();

double\*\* nmatr = get\_triangle\_matrix(matrix, size\_slau);

//print\_m(nmatr, size\_slau);

if (is\_it\_has\_zero\_str(nmatr, size\_slau))

{

double\* result = get\_result\_slau(nmatr, size\_slau);

print\_r(result, size\_slau);

delete result;

auto end = chrono::high\_resolution\_clock::now();

cout << "runtime == " << chrono::duration\_cast<chrono::microseconds>(end - beg).count();

}

else

{

cout << "Matrix is not union!"; // слау расходится или имеет бесконечно возможные решения

auto end = chrono::high\_resolution\_clock::now();

cout << "runtime == " << chrono::duration\_cast<chrono::microseconds>(end - beg).count();

}

for (long i = 0; i < size\_slau; i++)

{

delete[] matrix[i];

}

delete[] matrix;

}

**Тесты на результат:**

**Таблица 1: Тестирование алгоритма**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные параметры | Результат | Ожидаемый результат |
| 1 | A1 = { {2, -1, -6},{7, -4, 2},{1, -2, -4} }  B1 = {-3, 13, -1} | {1;-1;1} | {1;-1;1} |
| 2 | A2 = { {2, 5, 4, 1},{1, 3, 2, 1},{2, 10, 9, 7},{3, 8, 9, 2} }  B2 = {20,11,40,37} | {1;2;2,-0} | {1;2;2,0} |
| 3 | A3 = { {1, 2, 3},{2, -1, 2},{1, 1, 5} }  B3 = {1,6,-1} | {4, -0, -1} | {4, 0, -1} |

**Таблица 2: Замер времени с разным количеством неизвестных.**

|  |  |  |
| --- | --- | --- |
| Количество неизвестных\* | Без параллельности, мкс | C параллельностью, мкс |
| 100 | 66.656 | 61.195 |
| 1000 | 994.437 | 762.016 |
| 10000 | 314.295.265 | 208.719.107 |

**\***коэффициенты подбираются рандомно