Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский

политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3**

по дисциплине **«Технологии блокчейн и распределенные информационные системы»**

**Выполнил** студент гр. АСУ-20-1б

Чувашев Максим Алексеевич

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

*(подпись студента)*

**Проверил** доцент кафедры ИТАС

Щапов Владислав Алексеевич

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

*(оценка, дата, подпись преподавателя)*

Пермь, 2024

# 1 Анализ предметной области

## 1.1 Постановка задачи

С использованием языка C++ и библиотеки OpenMP разработать программу для решения СЛАУ методом Гаусса.

## 1.2 Инструменты разработки

Для выполнения лабораторной работы были использованы следующие инструменты:

1. редактор CLion;

# 2 Технология разработки

На листинге 1 представлена реализация «прямого хода» умножения СЛАУ методом Гаусса.

        // Прямой ход

        // Приведение системы к треугольному виду

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

            // Исключение xk из последующих уравнений

            // Выбирается опорный элемент

            double d = a[k \* n + k];

            #pragma omp for

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

                // Из уравнения i вычитается уравнение k

                double tmp  = a[i \* n + k] / d;

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

                    a[i \* n + j] -= tmp \* a[k \* n + j];

                b[i] -= tmp \* b[k];

            }

        }

Листинг 1 – «Прямой ход»

На листинге 2 представлена реализация «обратного хода» умножения СЛАУ методом Гаусса. Обратный ход реализован с использованием редукции.

// Обратных ход для системы треугольного вида

for (int k = n - 1; k >= 0; k--) {

sum = 0;

#pragma omp barrier

#pragma omp for reduction(+:sum)

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

sum += a[k \* n + i] \* x[i];

#pragma omp single

x[k] = (b[k] - sum) / a[k \* n + k];

}

Листинг 2 – «Обратный ход»

Полный код программы представлен в приложении A.

# 3 Результаты работы

Результаты выполнения задания лабораторной работы проиллюстрированы на рисунке 1.

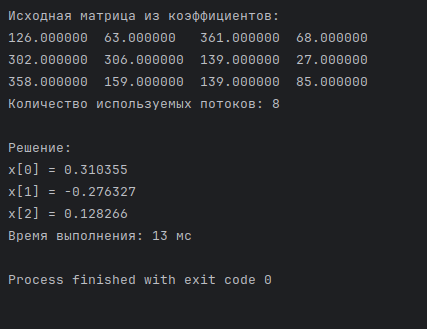


Рисунок 1 – Демонстрация работы

ПРИЛОЖЕНИЕ А

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <omp.h>

#include <chrono>

#include <iostream>

int n = 3;

double \*a = (double\*)malloc(sizeof(\*a) \* n \* n);

double \*b = (double\*)malloc(sizeof(\*b) \* n);

double \*x = (double\*)malloc(sizeof(\*x) \* n);

void MatrixInit() {

srand(time(NULL)); // seed the random number generator with the current time

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

b[i] = rand() % 100 + 1;

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

a[i \* n + j] = rand() % 500 + 1;

}

}

void PrintMatrix() {

printf("Исходная матрица из коэффициентов:\n");

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

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

printf("%f\t", a[i \* n + j]);

}

printf("%f", b[i]);

printf("\n");

}

}

void PrintResult() {

printf("\nРешение:\n");

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

printf("x[%d] = %f\n", i, x[i]);

}

}

int main() {

MatrixInit();

PrintMatrix();

double sum;

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

#pragma omp parallel

{

int num\_threads = omp\_get\_num\_threads();

#pragma omp single

{

printf("Количество используемых потоков: %d\n", num\_threads);

}

// Прямой ход

// Приведение системы к треугольному виду

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

// Исключение xk из последующих уравнений

// Выбирается опорный элемент

double d = a[k \* n + k];

#pragma omp for

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

// Из уравнения i вычитается уравнение k

double tmp = a[i \* n + k] / d;

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

a[i \* n + j] -= tmp \* a[k \* n + j];

b[i] -= tmp \* b[k];

}

}

// Обратных ход для системы треугольного вида

for (int k = n - 1; k >= 0; k--) {

sum = 0;

#pragma omp barrier

#pragma omp for reduction(+:sum)

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

sum += a[k \* n + i] \* x[i];

#pragma omp single

x[k] = (b[k] - sum) / a[k \* n + k];

}

}

auto t2 = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::milliseconds>(t2 - t1);

PrintResult();

// Вывод времени расчета решения

std::cout << "Время выполнения: " << duration.count() << " мс" << std::endl;

return 0;

}