Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ОТЧЕТ

к лабораторной работе №3

на тему

**ПРОГРАММИРОВАНИЕ**

**МНОГОЯДЕРНЫХ**

**АРХИТЕКТУР**

Выполнил:

студент гр. 153503

Кончик Д.С.

Проверил:

Калиновская А.А.

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Приложение А (обязательное) Листинг кода 7](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Ознакомиться с интерфейсом *OpenMP*, изучить основные принципы и подходы к программированию простых многопоточных приложений с его использованием.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*OpenMP* – интерфейс прикладного программирования (*API*) для масштабируемых *SMP*-систем в модели общей памяти. Исполняемый процесс в памяти может состоять из множественных нитей, которые имеют общее адресное пространство, но разные потоки команд и раздельные стэки. В простейшем случае, процесс состоит из одной нити, выполняющую функцию *main*. Нити иногда называют также потоками, легковесными процессами, *LWP*. *OpenMP* основан на существовании множественных потоков в общедоступной памяти. Схема процесса представлена на рисунке 1.

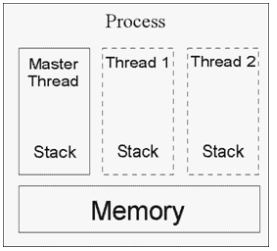


Рисунок 1

Все программы *OpenMP* начинаются как единственный процесс с главным потоком, который выполняется последовательно, пока не сталкиваются с первой областью параллельной конструкции. Создание нескольких потоков (*FORK*) и объединение (*JOIN*) находится на рисунке 2.

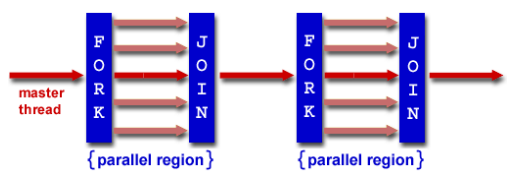


Рисунок 2

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В соответствии с вариантом №4 была реализована программа для решения системы линейных алгебраических уравнений методом Гаусса.

При запуске можно сделать выбор: ввести свою СЛАУ или же выбрать уже заготовленную большого размера (рисунок 1).

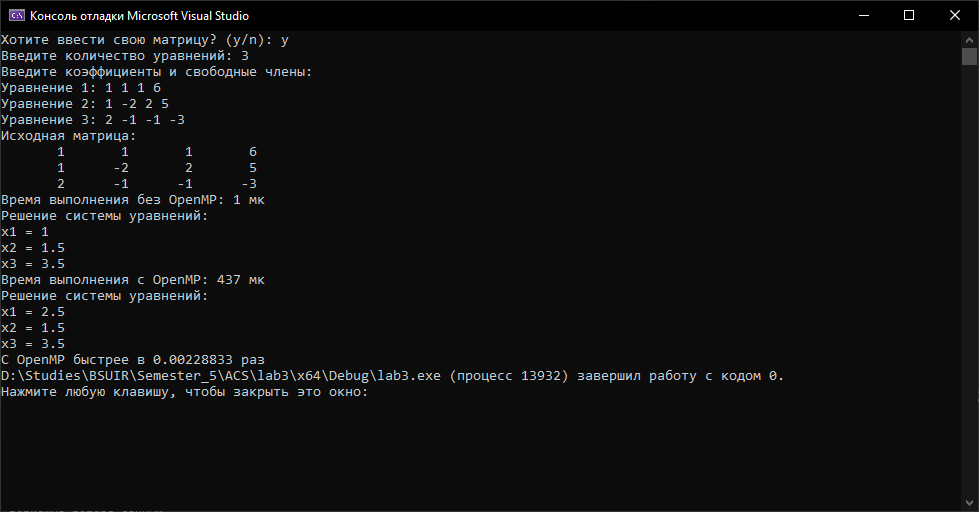


Рисунок 1 – Ввод своей СЛАУ

Также программа выводит время, затраченное на расчет без использования *OpenMP* и с его использованием. Однако разница не видна (или даже с процесс вычисления с распараллеливанием дольше) на маленьких матрицах ввиду накладных расходов на создание и управление потоками в *OpenMP*. Поэтому предлагается выбрать заготовленную СЛАУ на 1000 строк (рисунок 2).

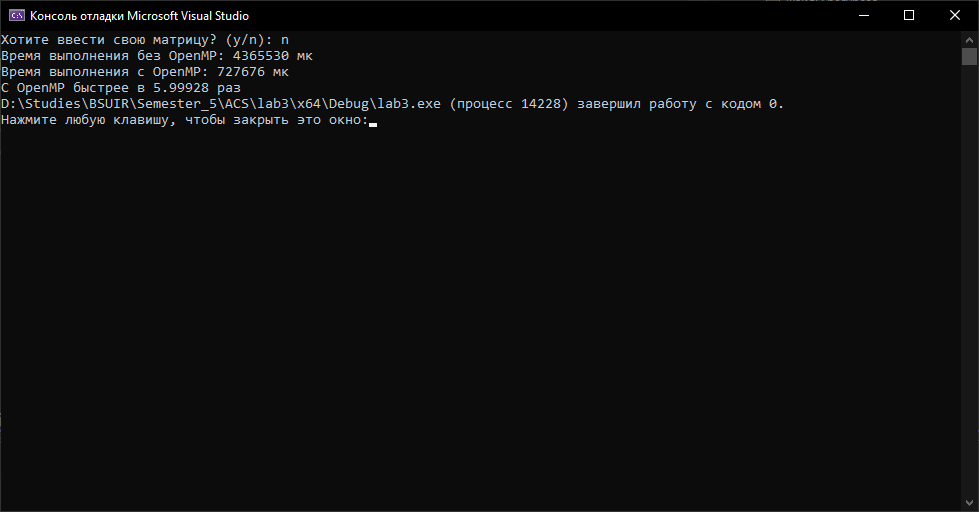


Рисунок 2 – Использование большой СЛАУ

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был изучен интерфейс *OpenMP* и основные принципы программирования многопоточных приложений. Была разработана программа для решения систем линейных алгебраических уравнений методом Гаусса с использованием *OpenMP*. Результаты тестирования программы показали, что на маленьких матрицах накладные расходы на создание и управление потоками в *OpenMP* могут превышать преимущества параллельного вычисления, поэтому для получения более точных результатов рекомендуется выбирать матрицы большего размера.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

#include <iostream>

#include <vector>

#include <iomanip>

#include <omp.h>

#include <chrono>

using namespace std;

using namespace chrono;

bool useReadyMadeMatrix = false;

// Функция для вывода матрицы

void printMatrix(const vector<vector<double>>& matrix) {

for (const auto& row : matrix) {

for (double element : row) {

cout << setw(8) << element;

}

cout << endl;

}

}

// Функция для выполнения метода Гаусса без OpenMP

int gaussianSolutionWithoutOpenMP(vector<vector<double>>& matrix) {

int rows = matrix.size();

int cols = matrix[0].size();

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

// Приведение матрицы к верхнетреугольному виду

for (int i = 0; i < rows - 1; i++) {

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

double factor = matrix[k][i] / matrix[i][i];

for (int j = i; j < cols; j++) {

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

}

}

}

// Обратный ход

vector<double> solution(rows, 0.0);

for (int i = rows - 1; i >= 0; i--) {

solution[i] = matrix[i][cols - 1] / matrix[i][i];

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

matrix[k][cols - 1] -= matrix[k][i] \* solution[i];

}

}

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

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Время выполнения без OpenMP: " << duration.count() << " мк" << endl;

// Вывод решения

if (!useReadyMadeMatrix) {

cout << "Решение системы уравнений:" << endl;

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

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

}

}

return duration.count();

}

// Функция для выполнения метода Гаусса с OpenMP

int gaussianSolutionWithOpenMP(vector<vector<double>>& matrix) {

int rows = matrix.size();

int cols = matrix[0].size();

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

// Приведение матрицы к верхнетреугольному виду

#pragma omp parallel for schedule(dynamic)

for (int i = 0; i < rows - 1; i++) {

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

double factor = matrix[k][i] / matrix[i][i];

for (int j = i; j < cols; j++) {

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

}

}

}

// Обратный ход

vector<double> solution(rows, 0.0);

#pragma omp parallel for schedule(dynamic)

for (int i = rows - 1; i >= 0; i--) {

solution[i] = matrix[i][cols - 1] / matrix[i][i];

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

matrix[k][cols - 1] -= matrix[k][i] \* solution[i];

}

}

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

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Время выполнения с OpenMP: " << duration.count() << " мк" << endl;

// Вывод решения

if (!useReadyMadeMatrix) {

cout << "Решение системы уравнений:" << endl;

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

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

}

}

return duration.count();

}

int main() {

setlocale(LC\_ALL, "Ru");

vector<vector<double>> matrix;

char choice;

cout << "Хотите ввести свою матрицу? (y/n): ";

cin >> choice;

useReadyMadeMatrix = !(choice == 'y' || choice == 'Y');

if (!useReadyMadeMatrix) {

cout << "Введите количество уравнений: ";

int n = 0; cin >> n;

// Создание матрицы

matrix = vector<vector<double>>(n, vector<double>(n + 1, 0.0));

// Ввод коэффициентов и свободных членов

cout << "Введите коэффициенты и свободные члены:" << endl;

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

cout << "Уравнение " << i + 1 << ": ";

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

cin >> matrix[i][j];

}

}

}

else {

// Использование заготовленной матрицы

matrix = vector<vector<double>>(1000, vector<double>(1000 + 1, 0.0));

}

// Вывод исходной матрицы

if (!useReadyMadeMatrix) {

cout << "Исходная матрица:" << endl;

printMatrix(matrix);

}

auto matrix1 = matrix;

auto matrix2 = matrix;

// Решение системы методом Гаусса без OpenMP

double withoutOpenMP = gaussianSolutionWithoutOpenMP(matrix1);

// Решение системы методом Гаусса с OpenMP

double withOpenMP = gaussianSolutionWithOpenMP(matrix2);

cout << "С OpenMP быстрее в " << withoutOpenMP / withOpenMP << " раз";

return 0;

}