Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

**Лабораторная работа №1:**

**«Цена вызова»**

Антон Гатченко Б22-525

2025 г.

*Используемая рабочая среда:*

* Процессор - AMD Ryzen 5 5600H (laptop), 6c/12t
* Оперативная память – DDR4 16 ГБ
* ОС - Windows 10 Pro 22H2 19045.4780, 64 bit
* IDE – GCC/G++ 13.1, OpenMP 201511

*Ход работы:*

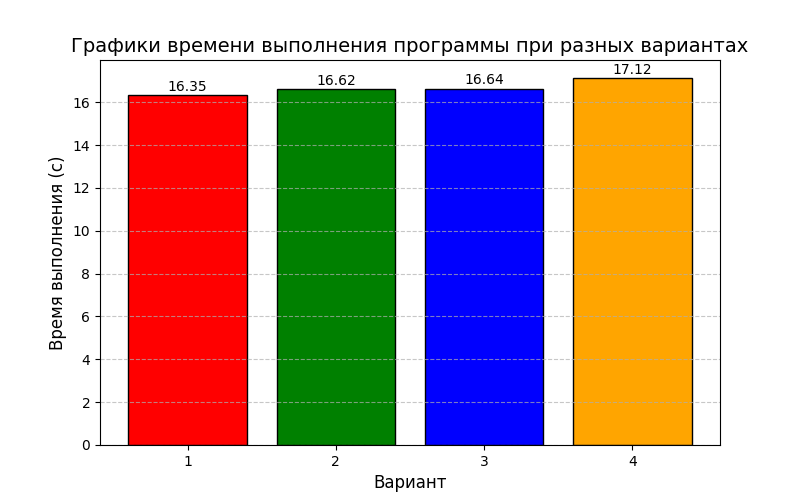
Для написания программы использовался язык C++.

Замеры времени выполнения всех вариантов программы проводились на уравнениях, большее количество не удалось поставить из-за недостатка оперативной памяти. Каждый коэффициент уравнения был задан типом float, значение варьировалось от до . Коэффициенты уравнения генерировались случайно, но итоговый набор был одинаковым для всех вариантов программы.

При компиляции программы флаги оптимизации *не* выставлялись.

Для 1 и 2 варианта код реализовывался в одном файле main.cpp; для 3 и 4 – в трёх файлах, main.cpp, equation\_solver.cpp, equation\_solver.h.

*График времени выполнения программы при разных вариантах:*



(1 – обычный, 2 – inline, 3 – две разных компиляции, 4 – передача обоих названий компилятору)

*Заключение:*

В ходе данной лабораторной работы было протестировано влияние подхода к компоновке на время работы программы.

Все варианты показали довольно схожие результаты, максимальное отличие оказалось между 1 (все функции в одном файле, без inline) и 4 (функции в разных файлах, компилятору переданы сразу все имена файлов) вариантами – секунды или , с. в 1 варианте и с. во втором

Варианты 2 (все функции в одном файле, с inline) и 3 (скомпилированы оба файла и из них скомпонована программа) показали практически идентичные результаты, и секунды соответственно.

Малое отличие между вариантами может быть вызвано компилятором, который может вносить свои изменения и оптимизации. Лучшая производительность достигается, когда весь код находится в одном файле, так как компилятор может применять глобальные оптимизации, однако современные компиляторы хорошо оптимизируют даже код в разных файлах, снижая влияние подхода к компоновке на время работы программы.

*Приложение:*

1. Исходный код программы с измерением времени работы программы:

#include <iostream>  
#include <cmath>  
#include <vector>  
#include <random>  
#include <chrono>  
#include <immintrin.h>  
#include <omp.h>  
  
#define SIZE 3e8  
#define SEED 15032025  
  
using std::vector, std::array, std::cout, std::endl;  
  
std::vector<std::array<double, 3>> generateRandomVectorsAVX(int count, int seed) {  
 std::cout << "Generating " << count << " random vectors" << std::endl;  
 const auto start = std::chrono::high\_resolution\_clock::now();  
  
 alignas(32) std::vector<std::array<double, 3>> result(count);  
 \_\_m256d scale = \_mm256\_set1\_pd(1e6);  
  
#pragma omp parallel  
 {  
 int thread\_id = omp\_get\_thread\_num();  
 std::mt19937 generator(seed + thread\_id);  
 std::uniform\_real\_distribution<double> distribution(-1e6, 1e6);  
  
#pragma omp for  
 for (int i = 0; i < count; ++i) {  
 \_\_m256d rnd = \_mm256\_set\_pd(  
 distribution(generator),  
 distribution(generator),  
 distribution(generator),  
 0.0 // Заполнитель для выравнивания  
 );  
 rnd = \_mm256\_mul\_pd(rnd, scale);  
 \_mm256\_store\_pd(result[i].data(), rnd);  
 }  
 }  
  
 const auto end = std::chrono::high\_resolution\_clock::now();  
 const std::chrono::duration<double> elapsed = end - start;  
 std::cout << "Finished generating after " << elapsed.count() << " seconds" << std::endl;  
  
 return result;  
}  
  
double calculateDiscriminant(double a, double b, double c){  
 return b \* b - 4 \* a \* c;  
}  
  
double calculateRoot1(double a, double b, double discriminant){  
 if (discriminant >= 0){  
 return (-b + sqrt(discriminant)) / (2 \* a);  
 }  
 return nan("");  
}  
  
double calculateRoot2(double a, double b, double discriminant){  
 if (discriminant >= 0){  
 return (-b - sqrt(discriminant)) / (2 \* a);  
 }  
 return nan("");  
}  
  
void timeCode(){  
 vector<array<double, 3>> equations = generateRandomVectorsAVX(SIZE, SEED);  
 const auto start = std::chrono::high\_resolution\_clock::now();  
  
 for (const auto &eq: equations){  
 const double a = eq[0];  
 const double b = eq[1];  
 const double c = eq[2];  
  
 const double discriminant = calculateDiscriminant(a, b, c);  
 const double root1 = calculateRoot1(a, b, discriminant);  
 const double root2 = calculateRoot2(a, b, discriminant);  
  
 // cout << a << "x^2 + " << b << "x + " << c << " = 0" << endl;  
 // if (discriminant < 0){  
 // cout << "D < 0" << endl;  
 // } else{  
 // cout << "Roots: x1 = " << root1 << ", x2 = " << root2 << endl;  
 // }  
 // cout << endl;  
 }  
  
 const auto end = std::chrono::high\_resolution\_clock::now();  
 const std::chrono::duration<double> elapsed = end - start;  
 cout << "Elapsed time: " << elapsed.count() << " seconds" << endl;  
}  
  
int main(){  
 timeCode();  
 return 0;  
}

1. CMakeLists.txt для 3го варианта:

cmake\_minimum\_required(VERSION 3.29)  
  
project(EquationSolver)  
  
set(CMAKE\_CXX\_STANDARD 20)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
# Добавление флага для AVX  
set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -mavx")  
  
find\_package(OpenMP REQUIRED)  
if(OPENMP\_FOUND)  
 set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -fopenmp")  
endif()  
  
# Создание статической библиотеки из equation\_solver.cpp  
add\_library(equation\_solver STATIC equation\_solver.cpp)  
  
# Создание исполняемого файла main  
add\_executable(EquationSolver main.cpp)  
  
# Связывание библиотеки equation\_solver с исполняемым файлом  
target\_link\_libraries(EquationSolver PRIVATE equation\_solver)  
  
# Указание заголовочных файлов  
target\_include\_directories(EquationSolver PUBLIC ${CMAKE\_CURRENT\_SOURCE\_DIR})

1. CMakeLists.txt для 4го варианта:

cmake\_minimum\_required(VERSION 3.29)  
  
project(EquationSolver)  
  
set(CMAKE\_CXX\_STANDARD 20)  
set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)  
  
# Добавление флага для AVX  
set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -mavx")  
  
# Поиск OpenMP  
find\_package(OpenMP REQUIRED)  
if(OPENMP\_FOUND)  
 set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -fopenmp")  
endif()  
  
# Добавление исполняемого файла  
add\_executable(EquationSolver main.cpp equation\_solver.cpp)  
  
# Указание заголовочных файлов  
target\_include\_directories(EquationSolver PUBLIC ${CMAKE\_CURRENT\_SOURCE\_DIR})