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

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №8: «»

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 19045.5854, 64 bit
- IDE – Clang 20.1.5

Практическая часть:

Для таймирования сборки и выполнения программы использовалась программа для решения квадратных уравнений, написанная на C++. Она состоит из 3 файлов: `main.cpp`, `equation_solver.cpp`, `equation_solver.h`. Её исходный код расположен в Приложении 1. Время сборки усреднялось по 5 пробегам, время выполнения программы - по 3.

Таймирование было автоматизировано с помощью Powershell-скрипта. Его можно найти в Приложении 2.

Тестировались три уровня оптимизации компилятора - `-O0`, `-O2`, `-Oz`, а также три уровня оптимизации компоновщика - `no LTO`, `thin LTO`, `full LTO`. Использовался компилятор Clang версии 20.1.5.

Таблица 1: Время сборки (с)

Уровень оптимизации	no LTO	thin LTO	full LTO
-O0	1.88	1.97	1.96
-O2	2.09	2.18	2.17
-Oz	2.00	2.06	2.04

Таблица 2: Время выполнения программы (с)

Уровень оптимизации	no LTO	thin LTO	full LTO
-O0	5.74	5.58	5.44
-O2	3.39	3.30	3.42
-Oz	4.09	4.01	3.98

Таблица 3: Размер исполняемого файла (КБ)

Уровень оптимизации	no LTO	thin LTO	full LTO
-O0	291.00	289.00	288.00
-O2	277.00	276.50	276.50
-Oz	265.00	264.50	265.00

Заключение

В ходе выполнения лабораторной работы были исследованы влияния различных уровней оптимизации компилятора (`-O0`, `-O2`, `-Oz`) и режимов LTO (`no LTO`, `thin LTO`, `full LTO`) на время сборки, производительность программы и размер исполняемого файла.

LTO повлияло на время сборки незначительно, разница между отсутствием `LTO` и `thin/full LTO` примерно в 5%, для ключа `-Oz` - около 2%.

Производительность выросла на 3% и 5% для `thin LTO` и `full LTO` соответственно при флаге `-O0`. Для флагов `-O2` и `-Oz` изменения на том же уровне, а вариант с `full LTO` оказался чуть медленнее даже `No LTO` для `-O2`, однако это может быть обусловлено погрешностью измерений.

Размер исполняемого файла практически не менялся, разница около 1%.

В целом, `thin LTO` и `full LTO` показали очень близкие результаты по всем параметрам. Разница между всеми уровнями `LTO` в рамках данного теста мала.

Приложение

1. Исходный код программы

```
// main.cpp
#include <iostream>
#include <vector>
#include <array>
#include <random>
#include <chrono>
#include <immintrin.h>
#include <omp.h>
#include "equation_solver.h"

#define SIZE 3e8
#define SEED 15032025

using std::vector, std::array, std::cout, std::endl;

vector<array<double, 3>> generateRandomVectorsAVX(int count, int seed) {
    cout << "Generating " << count << " random vectors" << endl;
    const auto start = std::chrono::high_resolution_clock::now();

    alignas(32) vector<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);
        }
        cout << 1 << endl;
    }

    const auto end = std::chrono::high_resolution_clock::now();
    const std::chrono::duration<double> elapsed = end - start;
    cout << "Finished generating after " << elapsed.count() << " seconds" <<
endl;

    return result;
}

void timeCode(){
    vector<array<double, 3>> equations = generateRandomVectorsAVX(SIZE, SEED);
    cout << "cool" << endl;
    const auto start = std::chrono::high_resolution_clock::now();
    double sum = 0.0;


    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);

        if (!std::isnan(root1) && !std::isnan(root2) && std::isfinite(root1) &&
std::isfinite(root2)){
            sum += root1 - root2;
        } else{
            sum++;
        }
    }

    const auto end = std::chrono::high_resolution_clock::now();
    const std::chrono::duration<double> elapsed = end - start;
```

```
    cout << "Elapsed time: " << elapsed.count() << " seconds;" << " Sum: " << sum
<< endl;
}

int main(){
    timeCode();
    return 0;
}
```

```
// equation_solver.cpp
#include <iostream>
#include <cmath>
#include <vector>
#include "equation_solver.h"

using std::vector, std::array, std::cout, std::endl;

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("");
}
```

```
// equation_solver.h
#pragma once

double calculateDiscriminant(double a, double b, double c);
double calculateRoot1(double a, double b, double discriminant);
double calculateRoot2(double a, double b, double discriminant);
```

2. Скрипт для автоматизации таймирования (Powershell)

```
# Список уровней оптимизации
$OptimizationLevels = "-O0", "-O2", "-Oz"

# Список режимов LTO
$LtoModes = @(
    @{ Name = "no LTO"; Flags = "" }
    @{ Name = "thin LTO"; Flags = "-flto=thin" }
    @{ Name = "full LTO"; Flags = "-flto" }
)
```

```powershell
foreach ($Level in $OptimizationLevels) {
    foreach ($Lto in $LtoModes) {

        # --- Подготовка ---
        $ltoName = $Lto.Name
        $extraFlags = $Lto.Flags

        Write-Host "Собираю: [$Level] + [$ltoName]"

        if (Test-Path main.exe) {
            Remove-Item main.exe
        }

        # --- Формируем флаги сборки ---
        $flags = @("-fopenmp", "-mavx", "-fuse-ld=lld", "--std=c++20")

        if ($Level -ne "") { $flags += $Level }
        if ($extraFlags -ne "") { $flags += $extraFlags }

        # --- Сборка проекта ---
        $totalBuildTime = 0.0
        $buildStopwatch = [System.Diagnostics.Stopwatch]::StartNew()

        for ($i = 1; $i -le 5; $i++) {
            clang++ main.cpp equation_solver.cpp -o main.exe $flags
        }

        $buildStopwatch.Stop()
        $totalBuildTime = $buildStopwatch.Elapsed.TotalSeconds
        $averageBuildTime = $totalBuildTime / 5
        $formattedBuildTime = "{0:F2}" -f $averageBuildTime

        # --- Запуск программы ---
        $totalRunTime = 0.0

        for ($i = 1; $i -le 3; $i++) {
            $runStopwatch = [System.Diagnostics.Stopwatch]::StartNew()
            .\main.exe
            $runStopwatch.Stop()
            $totalRunTime += $runStopwatch.Elapsed.TotalSeconds
        }

        $averageRunTime = $totalRunTime / 3
        $formattedRunTime = "{0:F2}" -f $averageRunTime

        # --- Размер файла ---
        $fileInfo = Get-Item main.exe
        $fileSizeKB = $fileInfo.Length / 1KB
        $formattedFileSize = "{0:F2}" -f $fileSizeKB

        # --- Вывод результатов ---
        Write-Host "$Level $ltoName $formattedBuildTime $formattedRunTime
$formattedFileSize"
    }
}
```