МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Изучение оптимизирующего компилятора»

Студента 2 курса, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

- *Изучение основных функций оптимизирующего компилятора, и некоторых примеров оптимизирующих преобразований и уровней оптимизации.*
- *Получение базовых навыков работы с компилятором GCC.*
- *Исследование влияния оптимизационных настроек компилятора GCC на время исполнения программы.*

ЗАДАНИЕ

1. *Написать программу на языке C или C++, которая реализует алгоритм вычисления числа Пи метом Монте-Карло. Алгоритм состоит в следующем. Сначала в квадрат с центром в начале координат и со стороной два вписывается круг с единичным радиусом. Затем в этом квадрате случайным образом с равномерным распределением генерируются N точек. Точка может попасть в окружность или нет (условие попадания $x^2 + y^2 \leq 1$). Далее определяется число M точек, попавших в круг. При достаточно большом числе бросков N, по значениям M и N вычисляется число Пи:*

$$\pi \approx \frac{4M}{N}$$

2. *Проверить правильность работы программы на нескольких тестовых наборах входных данных.*
3. *Выбрать значение параметра N таким, чтобы время работы программы было порядка 30-60 секунд.*
4. *Программу скомпилировать компилятором GCC с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og под архитектуру процессора x86.*
5. *Для каждого из семи вариантов компиляции измерить время работы программы при нескольких значениях N.*
6. *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
    1) *Титульный лист.*
    2) *Цель лабораторной работы.*
    3) *Вариант задания.*
    4) *Графики зависимости времени выполнения программы с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og от параметра N.*
    5) *Полный компилируемый листинг реализованной программы и команды для ее компиляции.*
    6) *Вывод по результатам лабораторной работы.*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Запустил программу без флагов оптимизации и нашёл значение параметра N = 1*10^9, при котором время работы программы составляет 30-60 секунд.*
2. *Написал bash-скрипт для запуска программы при разных уровнях оптимизации и значениях параметра N в интервале от 1*10^9 до 2*10^9 с шагом 2*10^8. (См. раздел «Строки компиляции и запуска программы»)*
3. *Запустил скрипт, перенаправляя вывод в файл test.txt.*
4. *При помощи данных, полученных из test.txt, составил графики зависимости времени работы программы от значения параметра N для разных уровней оптимизации. (См. раздел «Результат измерения времени работы программы»)*
5. *С помощью команды ls -l сравнил размеры бинарных файлов.*

Строки компиляции и запуска программы

Скрипт тестирования

```
#!/bin/bash

function build {
    rm -r bin_$1 2> /dev/null
    mkdir bin_$1
    echo "cmake -B bin_$1 -S src -D$1=true"
    cmake -B bin_$1 -S src -D$1=true
    cmake --build bin_$1
}

function test {
    sync
    bin_$1/lab2 $2
}

for OptLevel in O0 O1 O2 O3 Os Ofast Og
do
    build $OptLevel
done

for (( i=0; i <= 5; i++ ))
do
    for OptLevel in O0 O1 O2 O3 Os Ofast Og
    do
        echo "Run test with parameters:" $OptLevel $(( 1000000000 +
200000000 * $i ))
        test $OptLevel $(( 1000000000 + 200000000 * $i ))
        echo
    done
done
```

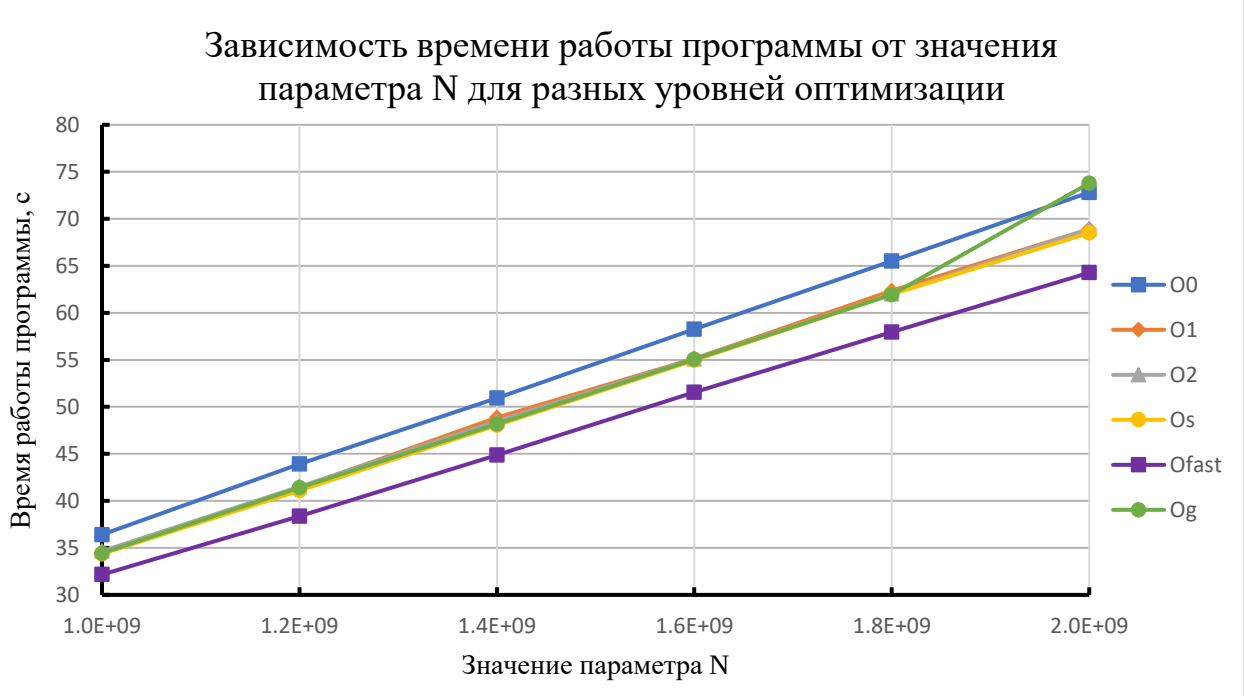
Команда для запуска скрипта тестирования

```
evmpu@comrade:~/21211/s.petrov1/lab2$ bash testing >> test.txt
```

Результат измерения времени работы программы

Таблица с полученными временем работы программы при разных уровнях оптимизации и значениях параметра N

		Уровень оптимизации						
		-O0	-O1	-O2	-O3	-Os	-Ofast	-Og
Значение параметра N	1E+09	36.4054	34.4521	34.6458	34.485	34.3604	32.1588	34.4363
	1.2E+09	43.929	41.4124	41.5084	41.3275	41.1124	38.3849	41.4188
	1.4E+09	50.92	48.8551	48.481	48.2309	48.034	44.8642	48.1635
	1.6E+09	58.2646	55.1189	55.098	55.357	54.9468	51.547	55.0399
	1.8E+09	65.5189	62.3497	61.9957	62.0034	61.9637	57.9421	61.9219
	2E+09	72.8102	68.8971	68.8909	69.0889	68.5096	64.2691	73.7889



Сравнение размеров бинарных файлов

```
evmpu@comrade:~/21211/s.petrov1/lab2$ ls -l bin*/lab2
-rwxrwxr-x 1 evmpu evmpu 17880 сен 14 21:34 bin_O0/lab2
-rwxrwxr-x 1 evmpu evmpu 17896 сен 14 21:34 bin_O1/lab2
-rwxrwxr-x 1 evmpu evmpu 17896 сен 14 21:34 bin_O2/lab2
-rwxrwxr-x 1 evmpu evmpu 17896 сен 14 21:34 bin_O3/lab2
-rwxrwxr-x 1 evmpu evmpu 19472 сен 14 21:34 bin_Ofast/lab2
-rwxrwxr-x 1 evmpu evmpu 17968 сен 14 21:34 bin_Og/lab2
-rwxrwxr-x 1 evmpu evmpu 17776 сен 14 21:34 bin_Os/lab2
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены:

- *Основные функции оптимизирующего компилятора, и некоторые примеры оптимизирующих преобразований и уровней оптимизации.*
- *Влияние оптимизационных настроек компилятора GCC на время исполнения программы и размер бинарного файла.*

По результатам проведённых исследований можно сделать следующие выводы:

- *Использование того или иного уровня оптимизации уменьшает время работы программы относительно уровня O0;*
- *Время работы, показанное программой при использовании флага -Ofast, оказалось наименьшим;*
- *Бинарный файл, сгенерированный при использовании флага -Os, имеет наименьший размер.*

ПРИЛОЖЕНИЕ 1 (ПОЛНЫЙ ЛИСТИНГ ПРОГРАММЫ)

src / main.cpp

```
#include <iostream>
#include <ctime>
#include "monte_carlo.h"

using namespace std;

int main(int argc, char **argv)
{
    if (argc == 1)
    {
        cerr << "No point count\n";
        return EXIT_FAILURE;
    }

    long long count = atoll(argv[1]);

    if (count < 0)
    {
        cerr << "Wrong point count\n";
        return EXIT_FAILURE;
    }

    struct timespec sysStart, sysEnd;
    clock_gettime(CLOCK_MONOTONIC_RAW, &sysStart);

    double pi = MonteCarloAlgorithm(count);

    clock_gettime(CLOCK_MONOTONIC_RAW, &sysEnd);
    double sysTime = sysEnd.tv_sec - sysStart.tv_sec + 1e-9 *
(sysEnd.tv_nsec - sysStart.tv_nsec);
    cout << "System time: " << sysTime << " sec.\n";

    cout << "PI: " << pi << "\n";

    return EXIT_SUCCESS;
}
```

src / CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.3)

set(CMAKE_CXX_COMPILER "/usr/bin/g++")

if(O0)
    message(STATUS "Flags -O0 included")
    set(CMAKE_CXX_FLAGS "-O0")
endif()
if(O1)
    message(STATUS "Flags -O1 included")
    set(CMAKE_CXX_FLAGS "-O1")
endif()
if(O2)
    message(STATUS "Flags -O2 included")
    set(CMAKE_CXX_FLAGS "-O2")
endif()
```

```
if(O3)
    message(STATUS "Flags -O3 included")
    set(CMAKE_CXX_FLAGS "-O3")
endif()
if(Os)
    message(STATUS "Flags -Os included")
    set(CMAKE_CXX_FLAGS "-Os")
endif()
if(Ofast)
    message(STATUS "Flags -Ofast included")
    set(CMAKE_CXX_FLAGS "-Ofast")
endif()
if(Og)
    message(STATUS "Flags -Og included")
    set(CMAKE_CXX_FLAGS "-Og")
endif()

project(lab2 CXX)

add_executable(lab2 main.cpp)
add_subdirectory(monte_carlo)
target_include_directories(lab2 PUBLIC monte_carlo)
target_link_libraries(lab2 PUBLIC monte_carlo)

find_library(LIBRT rt)
if(LIBRT)
    message(STATUS "Library rt installed")
    target_link_libraries(lab2 PUBLIC ${LIBRT})
else()
    message(STATUS "Library rt skipped")
endif()
```

src / monte_carlo / monte_carlo.cpp

```
#include "monte_carlo.h"

double MonteCarloAlgorithm(long long count)
{
    InitRand();

    double insideCount = 0.0;
    for (long long i = 0; i < count; ++i)
    {
        Point a;
        if (a.InsideCircle())
        {
            insideCount += 4.0;
        }
    }

    return insideCount / count;
}
```

src / monte_carlo / monte_carlo.h

```
#ifndef MONTE_CARLO_H
#define MONTE_CARLO_H
```

```
#include "point.h"
#include "random.h"

double MonteCarloAlgorithm(long long);

#endif // MONTE_CARLO_H
```

src / monte_carlo / CMakeLists.txt

```
add_library(monte_carlo STATIC monte_carlo.cpp)
target_include_directories(monte_carlo PUBLIC point)
add_subdirectory(point)
target_link_libraries(monte_carlo PUBLIC point)
```

src / monte_carlo / point / point.cpp

```
#include "point.h"

Point::Point()
{
    x = GenerateRand();
    y = GenerateRand();
}

bool Point::InsideCircle()
{
    return (x * x) + (y * y) <= 1.0;
}
```

src / monte_carlo / point / point.h

```
#ifndef POINT_H
#define POINT_H

#include "random.h"

class Point
{
public:
    Point();
    bool InsideCircle();
private:
    double x, y;
};

#endif // POINT_H
```

src / monte_carlo / point / CMakeLists.txt

```
add_library(point STATIC point.cpp)
target_include_directories(point PUBLIC random)
```

```
add_subdirectory(random)
target_link_libraries(point PUBLIC random)
```

src / monte_carlo / point / random / random.cpp

```
#include "random.h"

void InitRand()
{
    srand(time(NULL));
}

double GenerateRand()
{
    return (double)rand() / RAND_MAX;
}
```

src / monte_carlo / point / random / random.h

```
#ifndef RANDOM_H
#define RANDOM_H

#include <cstdlib>
#include <ctime>

void InitRand();
double GenerateRand();

#endif // RANDOM_H
```

src / monte_carlo / point / random / CMakeLists.txt

```
add_library(random STATIC random.cpp)
```