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

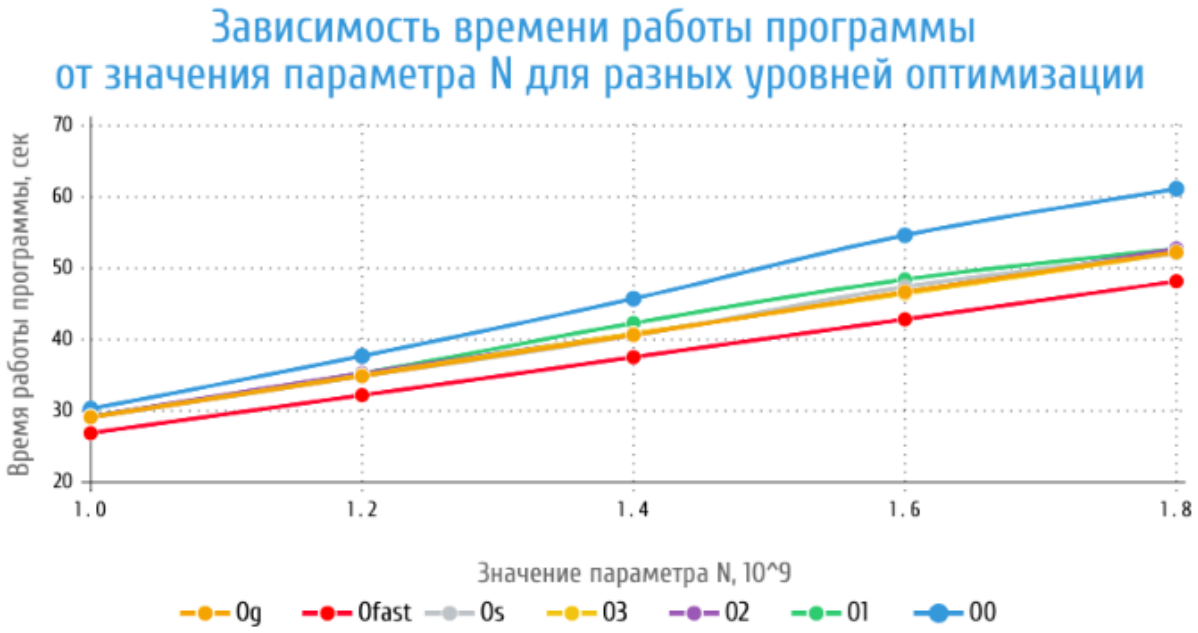
Скрипт тестирования

```
#!/bin/bash

function build {
    rm -r bin_$1 2> /dev/null
    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 <= 4; 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
```

Результат измерения времени работы программы

Результаты измерения времени

	O0	O1	O2	O3	Os	Ofast	Og
1.0E+09	30.3016	29.2299	29.2259	29.1448	29.0531	26.8813	29.117
1.2E+09	37.7028	35.2888	35.2792	35.0257	34.841	32.204	34.8802
1.4E+09	45.7469	42.3195	40.8204	40.8531	40.5766	37.5335	40.6627
1.6E+09	54.6318	48.4298	46.6138	46.454	47.4094	42.8552	46.6819
1.8E+09	61.1589	52.7679	52.7599	52.1996	52.1	48.177	52.2325



Сравнение размеров бинарных файлов

```
evmpu@comrade:~/21211/s.petrov1/lab2$ ls -l bin*/lab2
-rwxrwxr-x 1 evmpu evmpu 17568 сен 26 22:51 bin_O0/lab2
-rwxrwxr-x 1 evmpu evmpu 17576 сен 26 22:51 bin_O1/lab2
-rwxrwxr-x 1 evmpu evmpu 17576 сен 26 22:51 bin_O2/lab2
-rwxrwxr-x 1 evmpu evmpu 17576 сен 26 22:51 bin_O3/lab2
-rwxrwxr-x 1 evmpu evmpu 19160 сен 26 22:51 bin_Ofast/lab2
-rwxrwxr-x 1 evmpu evmpu 17648 сен 26 22:51 bin_Og/lab2
-rwxrwxr-x 1 evmpu evmpu 17456 сен 26 22:51 bin_Os/lab2
evmpu@comrade:~/21211/s.petrov1/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>

using namespace std;

double MonteCarloAlgorithm(long long count)
{
    srand(time(NULL));

    double insideCount = 0.0;
    for (long long i = 0; i < count; ++i)
    {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;

        if ((x * x) + (y * y) <= 1.0)
        {
            insideCount += 4.0;
        }
    }

    return insideCount / count;
}

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)

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