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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Зависимость времени работы программы от уровня оптимизации»

студента 2 курса, группы 20205

Муратова Максима Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент
Власенко А. Ю.

Новосибирск, 2021

СОДЕРЖАНИЕ

ЦЕЛИ

1. Измерить время работы программы, скомпилированной на разных уровнях компиляции;
2. Сравнить полученные данные между собой, особенно с компиляцией без оптимизации.

ЗАДАНИЕ

1. Цель программы – вычислить число Пи алгоритмом Монте-Карла: 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 ≤ 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. При таймера измерить время работы программы на компилятором G++ уровнях компиляции -O0, -O1, -O2, -O3, -Os, -Ofast, -Og под архитектуру процессора x86
3. Для всех семи уровней оптимизации измерить время работы программы при значениях ½ N, N, 3/2 N, где N – число точек, при котором программа будет работать от 30 до 60 секунд. Здесь
   $N = 2 * 10^9$

ОПИСАНИЕ РАБОТЫ

В качестве таймера была использована утилита **time**. Данная утилита выводит время работы программы по трём показателям:

1. User time (user) – время, которое работал пользовательский процесс (кроме времени работы других процессов);
2. Real time (real) – общее время работы программы согласно системному таймеру;
3. System time (sys) – время, затраченное на выполнение системных вызовов программы.

Как следует из определений, в качестве работы программы было выбранно именно user time, так как тут не учитывается время работы сторонних программ, тут только наша программа.

Исходный текст основной программы находится в файле `main.cpp`, см. Приложение 1.

Модуль тестирования времени вынесен в отдельную программу, исходный код которой содержится в файле `timetester.cpp`, см. Приложение 2. Это было сделано с целью автоматизации многократного исполнения (по 6 раз на каждую комбинацию уровня оптимизации с входными данными), из которых потом выбиралось наименьшее время.

Уровни оптимизации

- **O0**. Почти все оптимизации отключены, но время компиляции меньше, чем при использовании других уровней оптимизации. По умолчанию программы оптимизируются именно с ключом **O0**
- **O1**. Включены оптимизации для уменьшения размера бинарного исполняемого файла и такие оптимизации, уменьшающие время работы программы, которые не сильно замедляют работу компилятора.
- **O2**. Включены практически все доступные оптимизации, кроме тех, что ускоряют вычисления за счет увеличения размера кода.
- **O3**. Включены все оптимизации из уровня **O2**, к ним добавлены оптимизации времени работы программы, которые могут приводить к увеличению размера бинарного исполняемого файла.
- **Os**. Служит для оптимизации размера программ, в него включено подмножество оптимизаций из уровня **O2**.

- **Ofast**. Включает все оптимизации уровня **O3**, а также ряд других, таких как использование более быстрых и менее точных математических функций.
- **Og**. Производит все оптимизации, которые сохраняют возможность просмотра стека вызовов, фрагментов исходного текста программы, относящихся к разным уровням этого стека, и возможность приостановки программы для каждой строки исходного текста, содержащей операторы. Используется, следовательно, для отладки программы.

Некоторые виды оптимизаций

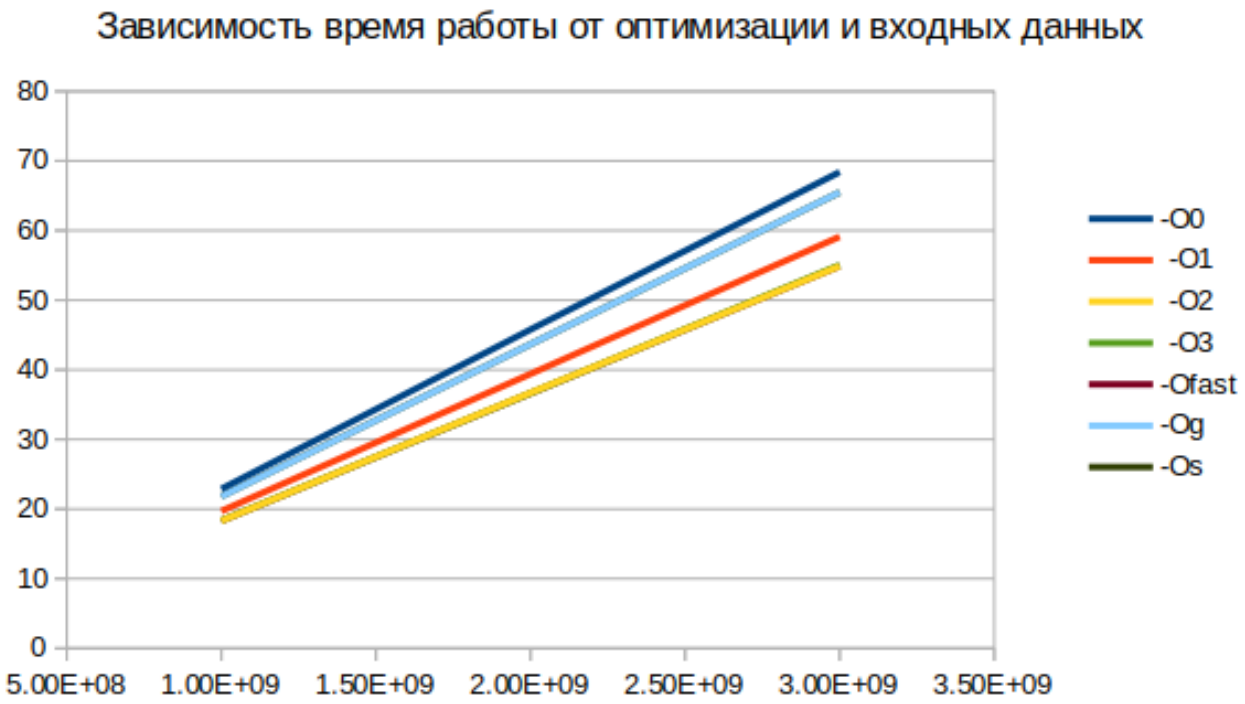
- **Удаление мёртвого кода** – удаление участка кода, который никак не влияет на результат программы. К мёртвому коду относится также код, которых никогда не исполняется. Подключается следующими ключами:
  - -fdce – удаление мёртвого кода, подключён везде, кроме на O0.
  - -fdse – удаление неиспользуемых переменных, подключён везде, кроме на O0.
  - -ftree-dce – удаление мёртвого кода в деревьях, подключён везде, кроме на O0.
  - -ftree-builtin-call-dce – удаление мёртвого кода в условном операторе, подключён с уровня O2.
- **Раскрутка циклов.** Исходный цикл преобразуется в другой цикл, в котором одно тело цикла содержит несколько тел старого цикла. При этом счётчик цикла меняется соответственно. Ключи раскрутки, оба нужно подключать отдельно, независимо от основного ключа оптимизации:
  - -funroll-loops – раскрутка циклов, количество итераций которых известно на этапе компиляции.
  - funroll-all-loops – раскрутка всех циклов, в том числе тех, у которых число итераций неизвестно на момент входа в цикл.
- **Встраивание функций (inline)**. Функция не вызывается, а вставляется в текст основной программы. За счёт размера исполняемого файла устраняется потребность в вызове функции и передачи параметров. Ключи:
  - -finline-small-functions – встраивает те функции, размер которых не превышает размер вызывающей функции, включено на O2 и O3.
  - -finline-functions – встраивание более крупных функций, включено с O3.
  - -finline-functions-called-once – встраивание функций, вызываемых только один раз, не включено только на O0.

    - -findirect-inlining – встраивание функций, вызываемых не напрямую (через указатель на функцию), включено на O2 и O3.
- **Перепрыгивание переходов.** Если в программе имеется цепочка последовательных переходов (условных или безусловных), она заменяется на единственный переход, который ведет сразу в окончательный пункт назначения. Преобразование включается ключом -fcrossjumping и активно на уровнях O2 и O3.
- **Устранение несущественных проверок указателей на null**. Разыменование nullptr всегда приводит к Segmentation fault. Поэтому, если в коде встречается проверка указателя на null после обращения по этому адресу, то такая проверка из кода исключается, так как указатель заведомо не нулевой. Преобразование включается ключом -fdelete-null-pointer-checks. Преобразование активно на всех уровнях, включая O0.
- **Перестановка инструкций.** Смысл оптимизации – поменять местами инструкции так, чтобы не нарушить правильность вычислений и чтобы переупорядовачивание привело к ускорению программы. По умолчанию включён на уровняы O1, O2, O3.
- **Хранение переменных в регистре.** Регистр – самая быстрая память. Если переменная небольшая и/или нечасто используется, её можно сразу в регистр записать, без помощи стека. Выигрыш и по скорости, и по использованию стека. Применяется с уровня O1.

В результате были получены следующие данные, в секундах:
N = 1 000 000 000

Уровень оптимизации	0,5 N	1,0 N	1,5 N
-O0	22.88	45.77	68.40
-O1	19.70	39.40	59.12
-O2	18.31	36.69	54.89
-O3	18.34	36.63	55.03
-Ofast	18.30	36.62	54.90
-Og	21.81	43.66	65.50
-Os	21.83	43.66	65.49

Данные точки образуют следующий график:



Легенда: Ox – число случайно сгенерированных точек, Oy – время выполнения программы в секундах

Есть совпавшие графики: Og и Os; O2, O3 и Ofast

На основании данного графика и табличных данных можно сделать следующие выводы:

1. Оптимизации уровня O3 не смогли ускорить выполнение программы, значит, они не сработали.

2. Сработали оптимизации уровней O1 и O2.
3. Самым быстрым по времени выполнения программы оказалась оптимизация O2, она в 1,25 быстрее оптимизации по умолчанию O0.

ЗАКЛЮЧЕНИЕ

1. Для данной программы самой эффективной оптимизацией оказался уровень O2, она примерно в 1,25 раза быстрее уровня оптимизации O0.
2. Сильно ускорить даже уровнем O2 программу не удалось.

Приложение 1. *main.cpp*

Основная программа, реализует предложенный алгоритм

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#define PREC 10000011

using namespace std;

int main(int argc, char **argv) {
    ofstream out("output.txt");
    if(argc != 2) {
        cout << "Wrong input\n";
        out << "0\n";
        return 0;
    }
    long long dotsCount = atoll(argv[1]), goodDots = 0;
    srand(time(NULL));

    for(long long iter = 0; iter < dotsCount; iter++) {
        long long x = rand() % PREC, y = rand() % PREC;
        if(x * x + y * y <= (PREC - 1) * (PREC - 1))
        goodDots++;
    }
    out.precision(10);
    out << 4. * goodDots / dotsCount << '\n';
    out.close();
    return 0;
}
```

Приложение 2. *timetester.cpp*

Программа, которая фиксирует время работы программы

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <string>
#include <cstdlib>

using namespace std;

int main(int argc, char **argv) {
    if(argc < 2) {
        cout << "no executable file\n";
        return 0;
    }
    char exeName[100] =
            "/home/evmpu/20205/Muratov/evmpu-lab1/";
    strcat(exeName, argv[1]);
    ifstream f(exeName);

    if(!f.is_open()) {
        cout << "the file doesn't exist\n";
        return 0;
    }
    f.close();

    ifstream ctrlFile("control.txt");
    double controlValue;
    ctrlFile >> controlValue;
    ctrlFile.close();
```

```
    char cmd[200];
    strcpy(cmd, "time -p -o report.txt -a ");
    strcat(cmd, exeName);
    for(int index = 2; index < argc; index++) {
        strcat(cmd, " ");
        strcat(cmd, argv[index]);
    }

    for(int iter = 0; iter < 6; iter++) {
        system(cmd);
        ifstream result("output.txt");
        double evaluated;
        result >> evaluated;
        cout << evaluated;
        string deviation = "echo " + to_string(evaluated /
                controlValue - 1) + " >> report.txt";
        system(deviation.c_str());
        result.close();
        system("echo \"\" >> report.txt");
    }
    system("echo \"\" >> report.txt");
    return 0;
}
```

Приложение 3. *createExes.sh*

Скрипт генерации программ

```
#!/bin/bash
g++ main.cpp -O0 -o exe0
g++ main.cpp -O1 -o exe1
g++ main.cpp -O2 -o exe2
g++ main.cpp -O3 -o exe3
g++ main.cpp -Os -o exes
g++ main.cpp -Ofast -o exefast
g++ main.cpp -Og -o exeg
```

- Компиляция программы-замерщика:
  $ g++ timetester.cpp –o timetester
- Комлиляция основной программы для всех уровней компиляции
  $ ./createExes.sh
- Тест исполняемого файла. Результаты сохраняются в `report.txt`
  $ ./timetester prog M
  # M – число «бросков» точек
  #prog – одна из исполняемых программ

- Справочник по ключам оптимизации компилятора g++
https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html