**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**

«*ИЗУЧЕНИЕ ОПТИМИЗИРУЮЩЕГО КОМПИЛЯТОРА*»

студента 2 курса, группы 22204

**Мустакаева Дмитрия Дмитриевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

Андрей Юрьевич Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_gjdgxs)

[ЗАДАНИЕ 3](#_30j0zll)

[ОПИСАНИЕ РАБОТЫ 4](#_1fob9te)

[ЗАКЛЮЧЕНИЕ](#_2et92p0) 7

[Приложение 1. Полный компилируемый листинг](#_3znysh7) 8

Приложение 2. Bash скрипт для компиляции с различными уровнями оптимизациями 11

1. **ЦЕЛЬ**
2. Изучить основные функции оптимизирующего компилятора,
3. Исследовать влияние некоторых его настроек на время работы программы
4. Получить базовые навыки работы с компилятором GCC.

1. **ЗАДАНИЕ**

1.Написать программу на языке C или C++, содержащую функцию, которая реализует алгоритм bubble sort.

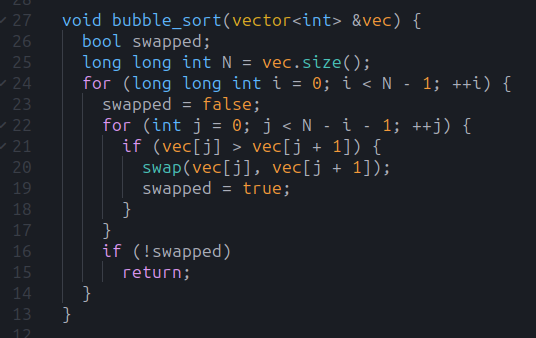
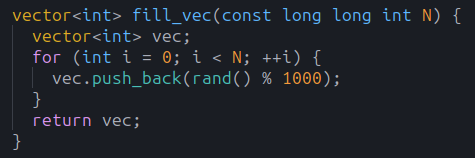
2.Проверить правильность работы программы на нескольких тестовых наборах данных

3. Выбрать значение параметра N таким, чтобы время работы программы было порядка 15 секунд

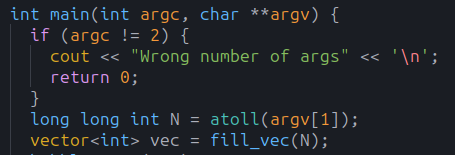
4. Скомпилировать программу компилятором GCC с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og.

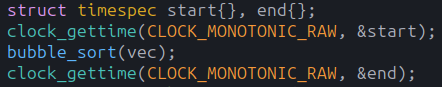
1. **ОПИСАНИЕ РАБОТЫ**

1. Написана программа, реализующая алгоритм пузырьковой сортировки, сам алгоритм выведен в отдельную функцию:

  
написана функция для заполнения вектора случайными числами:  


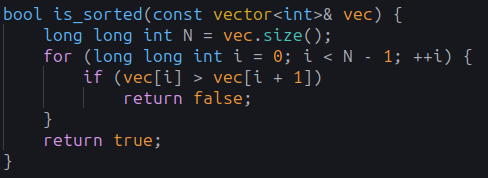
а также реализован ввод параметра N - (кол-во элементов в векторе) через аргумент командной строки:

  
2. Для измерения времени работы функции использовалась библиотечная функция *clock\_gettime:*

**Функция clock\_gettime с параметром CLOCK\_MONOTONIC является одним из способов получения текущего времени в системе в языке программирования C/C++. Она предоставляет доступ к монотонному таймеру, который предназначен для измерения времени без учета изменений в системном времени, таких как смена временных зон или коррекция времени.

В данном случае сохраняется значение таймера перед выполнением алгоритма сортировки и после него.

3. Проверена корректность сортировки, при помощи функции is\_sorted

  
4. Путём подбора выбрано значение N, при котором время работы функции составляет порядка 30-60 секунд:

| Значение N | Время работы функции, с. |
| --- | --- |
| 100000 | 48.056 |

5. Программа скомпилирована компилятором GCC с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og под архитектуру процессора x86 (x86-64).

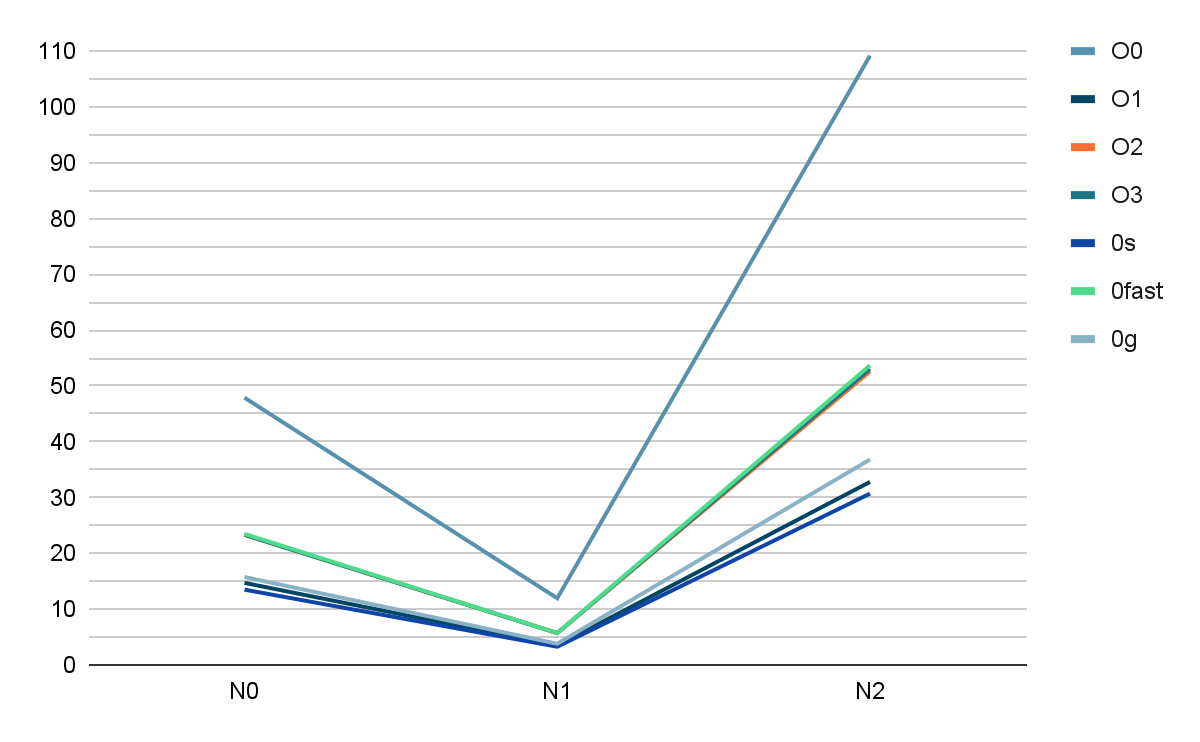
6. Составлена таблица с результатами измерения времени работы программы при различных значениях параметра N с различными уровнями оптимизации. Значения параметра N выбрал следующие:

**N0 = 100000,**

**N1 = 0.5N = 50000,**

**N2 = 1.5N = 150000**.В первом столбце таблицы приведены параметры компиляции, в первом ряду - значения параметра N. Время работы функции в секундах записано на пересечении соответствующих столбца параметра N и строки параметра компиляции.

|  | N0 | N1 | N2 |
| --- | --- | --- | --- |
| -O0 | 47.8919 | 11.944 | 109.182 |
| -O1 | 14.6986 | 3.47491 | 32.7924 |
| -O2 | 23.3237 | 5.70009 | 52.5771 |
| -O3 | 23.3109 | 5.495 | 53.0274 |
| -Os | 13.4883 | 3.25949 | 30.6943 |
| -Ofast | 23.4829 | 5.70944 | 53.6806 |
| -Og | 15.7547 | 3.79306 | 36.8079 |

7. По полученной таблице построена диаграмма зависимости времени выполнения программы с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og от параметра N:  


**ЗАКЛЮЧЕНИЕ**

В процессе работы были изучены подходы к измерению времени работы программы.

Была написана программа на языке C++, в которой были применен метод замера времени работы программы clock\_gettime - позволяет измерить время работы подпрограммы, вне зависимости от сторонних процессов.

Также были изучены и применены на практике различные флаги компиляции, позволяющие сократить время исполнения программы

Программа была протестирована при различных входных данных и уровнях оптимизации. В итоге, было выявлено: подпрограмма выполняется

значительно быстрее с уровнем оптимизации программы –O1, чем с –O0, однако переход с -O1 на последующие уровни оптимизации не всегда сокращал время исполнения, а иногда даже увеличивал его.

1. **Приложение 1**  *Полный компилируемый листинг*

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <vector>

**using** **namespace** std;

vector<**int**> fill\_vec(**const** **long** **long** **int** N) {

vector<**int**> vec;

**for** (**int** i = 0; i < N; ++i) {

vec.push\_back(rand() % 1000);

}

**return** vec;

}

**void** print\_vec(**const** vector<**int**>& vec) {

**long** **long** **int** N = vec.size();

**for** (**long** **long** **int** i = 0; i < N; ++i) {

cout << vec[i] << " ";

}

cout << endl;

}

**bool** is\_sorted(**const** vector<**int**>& vec) {

**long** **long** **int** N = vec.size();

**for** (**long** **long** **int** i = 0; i < N - 1; ++i) {

**if** (vec[i] > vec[i + 1])

**return** **false**;

}

**return** **true**;

}

**void** bubble\_sort(vector<**int**>& vec) {

**bool** swapped;

**long** **long** **int** N = vec.size();

**for** (**long** **long** **int** i = 0; i < N - 1; ++i) {

swapped = **false**;

**for** (**int** j = 0; j < N - i - 1; ++j) {

**if** (vec[j] > vec[j + 1]) {

swap(vec[j], vec[j + 1]);

swapped = **true**;

}

}

**if** (!swapped)

**return**;

}

}

**int** main(**int** argc, **char**\*\* argv) {

**if** (argc != 2) {

cout << "Wrong number of args" << '\n';

**return** 0;

}

**long** **long** **int** N = atoll(argv[1]);

srand(0);

vector<**int**> vec = fill\_vec(N);

// print\_vec(vec);

**struct** timespec start {

}, end{};

// cout << "Before sort: " << is\_sorted(vec) << '\n';

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

bubble\_sort(vec);

// cout << "After sort: " << is\_sorted(vec) << "\n";

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

cout << "Time taken: " << end.tv\_sec - start.tv\_sec + 0.000000001 \* (end.tv\_nsec - start.tv\_nsec) << endl;

// print\_vec(vec);

**return** 0;

}

1. **Приложение 2**  *Bash скрипт для компиляции с различными уровнями оптимизациями*

#!/bin/bash

mkdir -p o\_builds && cd o\_builds

g++ -O0 ../src/main.cpp -o task1\_O0

g++ -O1 ../src/main.cpp -o task1\_O1

g++ -O2 ../src/main.cpp -o task1\_O2

g++ -O3 ../src/main.cpp -o task1\_O3

g++ -Os ../src/main.cpp -o task1\_Os

g++ -Ofast ../src/main.cpp -o task1\_Ofast

g++ -Og ../src/main.cpp -o task1\_Og

Команды для запуска программы:  
./task1\_O0 N  
./task1\_O1 N  
./task1\_O2 N  
./task1\_O3 N  
./task1\_Os N  
./task1\_Ofast N  
./task1\_Og N  
Где N - вводимый параметр.