###### 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**

ОПРЕДЕЛЕНИЕ ВРЕМЕНИ ПРИКАЛДНЫХ ПРОГРАММ И ИЗУЧЕНИЕ ОПТИМИЗИРУЮЩЕГО КОМПИЛЯТОРА

студентки 2 курса, группы 21205

**Евдокимовой Дарьи Евгеньевны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук, доцент

А.Ю.Власенко

Новосибирск 2022

**СОДЕРЖАНИЕ**

[ЦЕЛЬ](#_heading=h.gjdgxs) **3**

[ЗАДАНИЕ](#_heading=h.30j0zll) **3**

[ОПИСАНИЕ РАБОТЫ](#_heading=h.1fob9te) **3**

[ЗАКЛЮЧЕНИЕ](#_heading=h.3znysh7) **7**

[ПРИЛОЖЕНИЕ 1. Вывод упрощенной формулы для разложения синуса в ряд Тейлора](#_heading=h.xgp9ve825qzn) **8**

[ПРИЛОЖЕНИЕ 2. Листинг программы с библиотечной функцией clock\_gettime](#_heading=h.3dy6vkm) **9**

# ЦЕЛЬ

1. Изучение основных функций оптимизирующего компилятора, и некоторых примеров оптимизирующих преобразований и уровней оптимизации.
2. Получение базовых навыков работы с компилятором GCC.
3. Исследование влияния оптимизационных настроек компилятора GCC

на время исполнения программы.

# ЗАДАНИЕ

Вариант задания: 4.

Написать программу на языке C или C++, содержащую функцию, которая реализует выбранный алгоритм из задания. Программа должна принимать значение N через параметр в командной строке.

Проверить правильность работы программы на нескольких тестовых наборах входных данных.

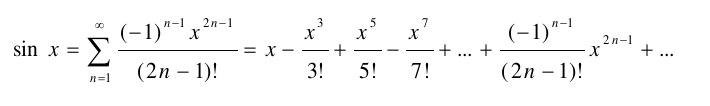
Скомпилировать программу компилятором GCC с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Og под архитектуру процессора x86 (x86-64).

# ОПИСАНИЕ РАБОТЫ

В ходе задания использовался компьютер с архитектурой amd64, с операционной системой Ubuntu 20.04.5 LTS и процессором Intel® Core™ i3-6100U CPU @ 2.30GHz × 4.

###### Пошаговое описание выполненной работы

1. Был создан файл pract1.c
2. Была написана компьютерная программа, которая вычисляет sin(x) с помощью разложения в степенной ряд по первым N членам этого ряда (см. «Рис.1»).

  
Рис.1. Разложение синуса в ряд Тейлора

1. Нетрудно заметить, что степень x в числителе увеличивается на 2 в каждом новом слагаемом, и в знаменателе число под знаком факториала также увеличивается на 2 в каждом новом слагаемом. По такому принципу запрограммируем разложение ряда. Вывод формула представлен в Приложении 1.
2. Код скомпилирован командой **gcc -o sinex.o pract1.c**,   
   где sinex.o - исполнительный файл.
3. Запуск программы производится с помощью команды

**./sinex.o argv[1] argv[2]**

На вход программы первый аргумент - число x (угол, синус которого необходимо посчитать), второй аргумент - количество членов n, по которым будет выполнено разложение. Листинг программы представлен в Приложении 3. На выходе ожидается одно число: значение синуса при разложении по ряду Тейлора.

Пусть угол, синус которого нужно посчитать, равен 30 градусам во всех запусках программ.

###### Описание методики для определения времени работы программы

Замеры времени работы программы будем проводить с помощью библиотечной функции clock\_gettime, которая получает значения системного таймера в ОС Linux/UNIX.

Функция clock\_gettime с параметром CLOCK\_MONOTONIC\_RAW сохраняет значение системного таймера в структуру struct timespec. Структура состоит из двух полей: tv\_sec и tv\_nsec (можно считать их тип long int), задающих количество секунд и наносекунд, прошедших с некоторого неспецифицированного момента времени в прошлом.

В приведённом коде программы (см.Приложение 3) сохраняется значение таймера перед выполнением некоторого кода и после него. Разница показаний преобразуется в секунды и выводится на экран. Реализация функции clock\_gettime находится в библиотеке rt, поэтому при компиляции программы необходимо добавить ключ компиляции ‘-lrt’.

###### Описание выполненной работы по выбранному методу измерения времени

1. Подберем такое значение n, при котором время работы программы будет оставлять 30-60 секунд. Подобранное значение: n = 5000000000.
2. Скомпилируем программу с уровнями оптимизации -O0, -O1, -O2, -O3, -Os,   
   -Ofast, -Og. Для каждого уровня оптимизации замерим время работы программы при подобранном значении n = 5000000000; 0,5n = 2500000000; 1,5n = 7500000000.

Команды компиляции и запуска отразим в таблице (см. Таблицу 1).

Табл1. Команды компиляции и запуска программы

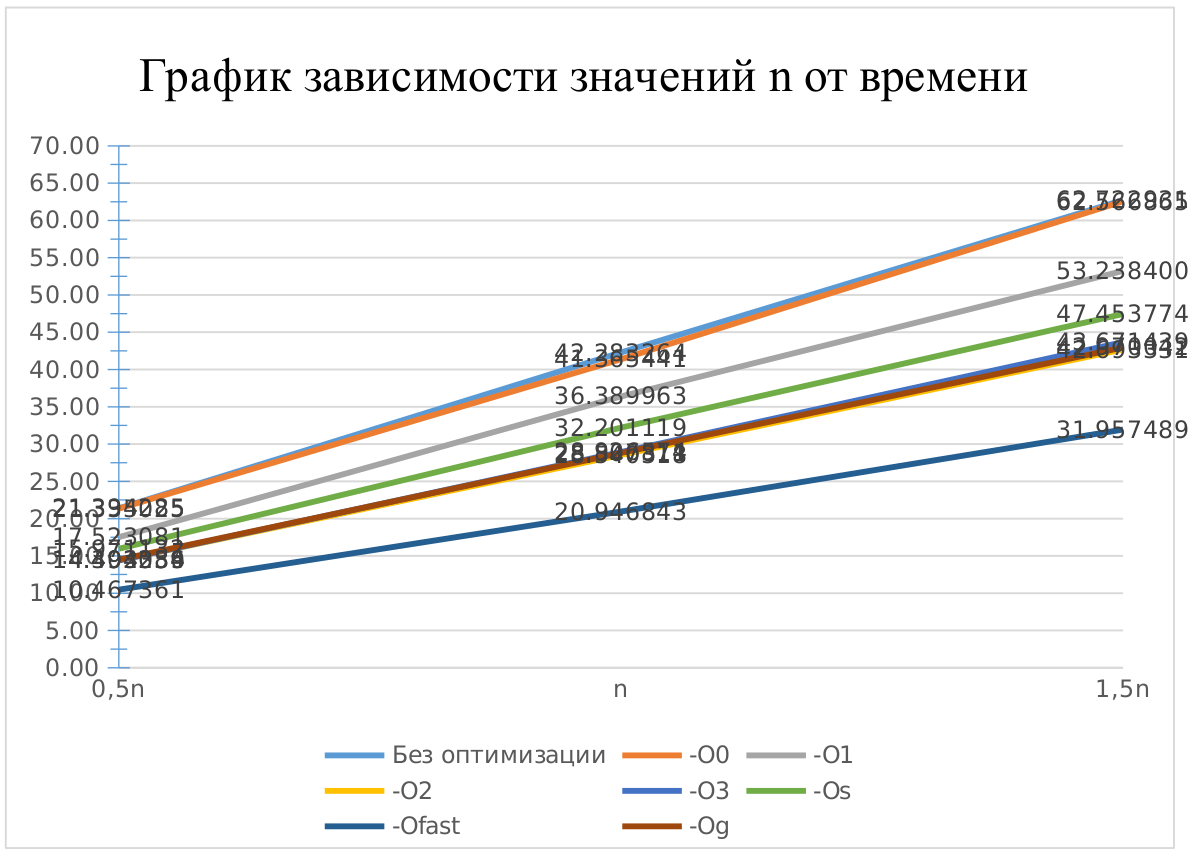
|  |  |  |
| --- | --- | --- |
| Уровень оптимизации | Строка компиляции | Строка запуска |
| -O0 | gcc -O0 pract1.c -o sin0.out | ./sin0.out5000000000 |
| -O1 | gcc –O1 pract1.c -o sin1.out | ./sin1.out 5000000000 |
| -O2 | gcc –O2 pract1.c -o sin2.out | ./sin2.out 5000000000 |
| -O3 | gcc –O3 pract1.c -o sin3.out | ./sin3.out 5000000000 |
| -Os | gcc -Os pract1.c -o sins.out | ./sins.out 5000000000 |
| -Ofast | gcc - Ofast pract1.c -o sinfast.out | ./sinfast.out 5000000000 |
| -Og | gcc -Og pract1.c -o sins.out | ./sing.out 5000000000 |

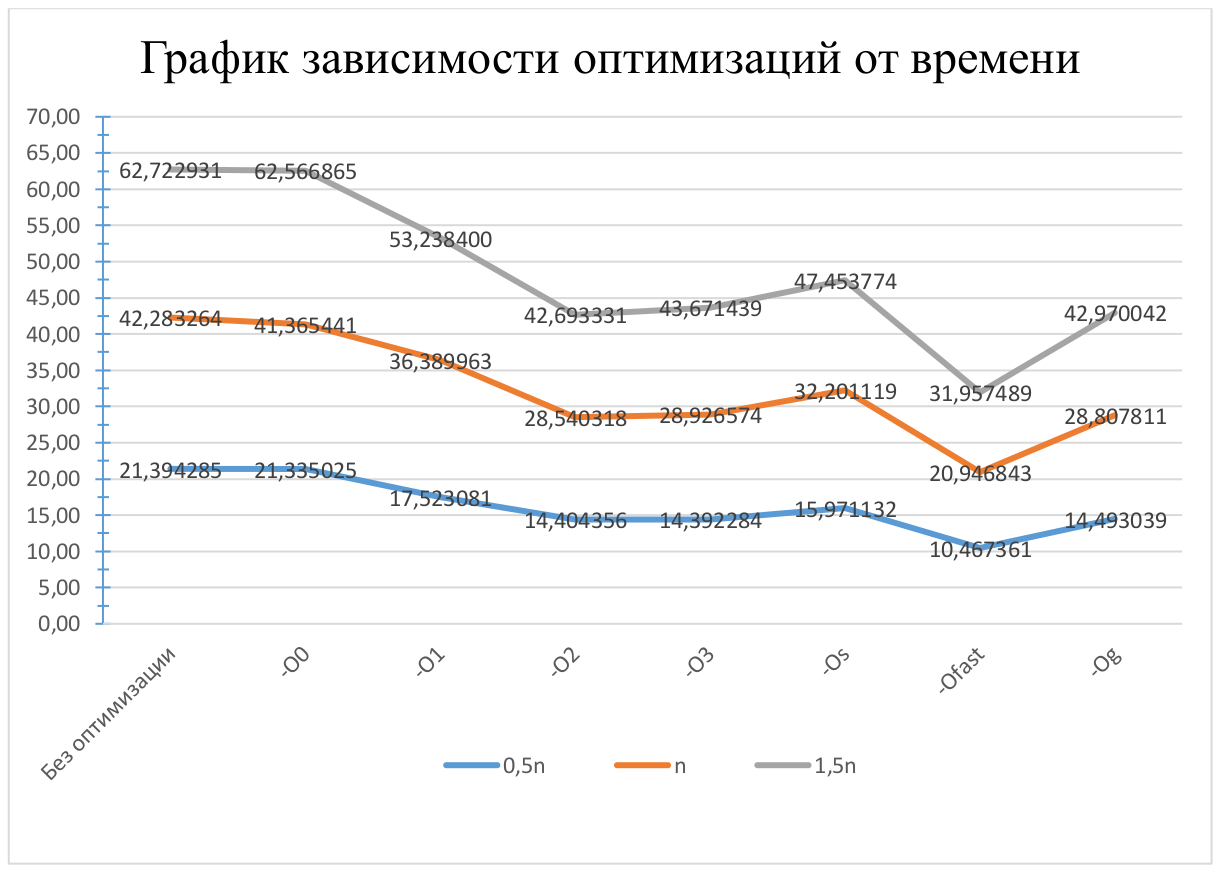
1. Добавим полученные результаты измерений в таблицу (см. Таблицу 2).

Табл2. Результаты измерений работы программы

|  |  |  |  |
| --- | --- | --- | --- |
| Уровень оптимизации | Время, сек | | |
| 0,5n | n | 1,5n |
| Без оптимизации | 21,394285 | 42,283264 | 62,722931 |
| -O0 | 21,335025 | 41,365441 | 62,566865 |
| -O1 | 17,523081 | 36,389963 | 53,238400 |
| -O2 | 14,404356 | 28,540318 | 42,693331 |
| -O3 | 14,392284 | 28,926574 | 43,671439 |
| -Os | 15,971132 | 32,201119 | 47,453774 |
| -Ofast | 10,467361 | 20,946843 | 31,957489 |
| -Og | 14,493039 | 28,807811 | 42,970042 |

1. Графическое представление результатов экспериментов приведено на графиках зависимости различных значений n от времени (см. Рис. 2) и зависимости различных оптимизаций от времени (см. Рис. 3).

  
Рис.2. График зависимости значений n от времени

  
Рис.3. График зависимости оптимизаций от времени от времени

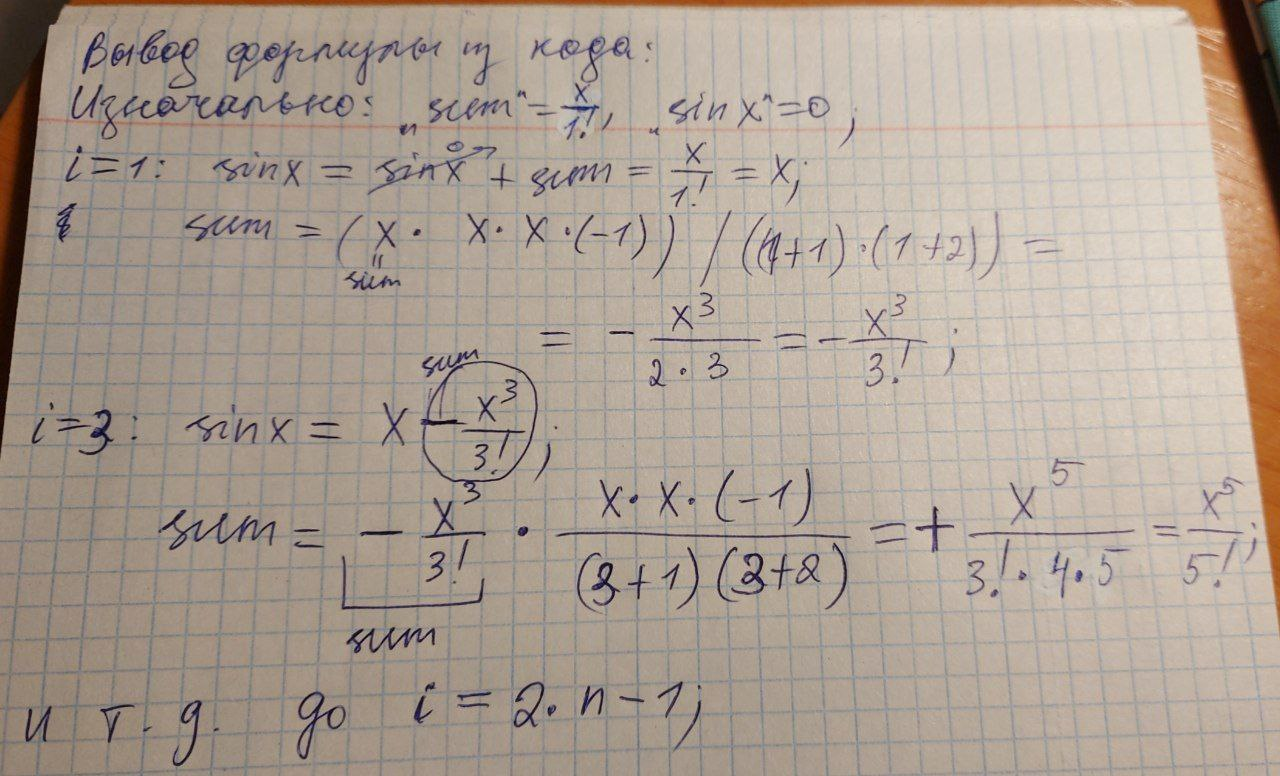
# ЗАКЛЮЧЕНИЕ

В нашей работе мы узнали базовые команды при работе с компилятором, изучили основные флаги для оптимизации и как они влияют на время работы программы.

По итогу работы были представлены различные способы оптимизации и определены зависимости этих оптимизаций от времени и n-членов, по которым было необходимо разложить синус в ряд Тейлора.

В ходе выполнения задания было выявлено, что код без оптимизаций и оптимизация -O0 работает почти одинаковое время; оптимизация -O1 работает быстрее -O1, но  
медленнее -Os; оптимизации -O2, -O3, -Og работают быстрее упомянутых выше. Различие во времени работы программы наблюдается в сотых и тысячных долях секунды.   
Оптимизация -Ofast показала себя быстрее всего.

# ПРИЛОЖЕНИЕ 1. Вывод упрощенной формулы для разложения синуса в ряд Тейлора



# ПРИЛОЖЕНИЕ 2. Листинг программы с библиотечной функцией clock\_gettime

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define PI 3.1415926535897

double CalcSin(double x, long long n){

    double sinx = 0;

    x = x \* PI / 180;

    double sum = x;

    for (long long i = 1; i <= 2 \* n - 1; i += 2){

        sinx += sum;

        sum = (sum \* x \* x \* (-1)) / ( (i + 1) \* (i + 2)) ;

    }

    return sinx;

}

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

    struct timespec start, end;

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

    if (argc == 1){

        printf("Bad input. Enter x and n in command line");

                return 0;

    }

    double x = atoll(argv[1]);

    long long n = atoll(argv[2]);

    double sinx = CalcSin(x, n);

    printf("%lf\n", sinx);

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

    printf("Time taken: %lf sec.\n", end.tv\_sec-start.tv\_sec

                + 0.000000001\*(end.tv\_nsec-start.tv\_nsec));

    return 0;

}

Строка компиляции: gcc –o sinex.o pract1.c -lrt

Строка запуска: ./sinex.o 30 5000000000