###### 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 методик измерения времени работы программы и оптимизирующих компиляторов»

студента 2 курса, группы 20203

**Синюкова Валерия Константиновича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель: доцент кафедры параллельных вычислений Власенко Андрей Юрьевич

Новосибирск 2021

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc82641778)

[ЗАДАНИЕ 3](#_Toc82641779)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc82641780)

[ЗАКЛЮЧЕНИЕ 5](#_Toc82641781)

[Приложение 1. *Листинг реализованной программы* 6](#_Toc82641782)

[Приложение 2. *Команды для компиляции и запуска программы* 7](#_Toc82641783)

# ЦЕЛЬ

1. Изучение методики измерения времени работы подпрограммы.

2. Изучение приемов повышения точности измерения времени работы подпрограммы.

3. Изучение способов измерения времени работы подпрограммы.

4. Измерение времени работы подпрограммы в прикладной программе.

5. Изучение основных функций оптимизирующего компилятора, и некоторых примеров оптимизирующих преобразований и уровней оптимизации.

6. Получение базовых навыков работы с компилятором GCC.

7. Исследование влияния оптимизационных настроек компилятора GCC на время исполнения программы.

# ЗАДАНИЕ

Формулировка общего задания:

1. 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.

2. Проверить правильность работы программы на нескольких тестовых наборах входных данных.

3. Выбрать значение параметра N0 таким, чтобы время работы функции было от 30 до 60 секунд.

4. Программу скомпилировать компилятором GCC с уровнями оптимизации -O0, -O1, -O2, -O3, -Os, -Ofast, -Ogпод архитектуру процессора x86 (x86-64).

5. Для каждого из семи вариантов компиляции измерить время работы программы при нескольких значениях N (0.5\* N0, N0, 1.5\* N0)**.**

6. Составить отчет по лабораторной работе.

Выбранный вариант:

1) Алгоритм вычисления числа Пи с помощью разложения в ряд (ряд Грегори-Лейбница) по формуле Лейбница N первых членов ряда:

# ОПИСАНИЕ РАБОТЫ

1. Была написана программа на языке C, содержащая функцию, которая реализует данный алгоритм из задания. Программа принимает на вход значение N через параметр командной строки. (N – количество членов ряда Грегори-Лейбница, которые будут суммироваться)
2. Программа была проверена на нескольких наборах тестовых данных:

|  |  |
| --- | --- |
| **N** | **Результат (π)** |
| 1 | 3.041840 |
| 10 | 3.131593 |
| 100 | 3.140593 |
| 1000 | 3.141493 |

1. С помощью утилиты time было измерено время выполнения программы, выбрано значение параметра N0 = 109, при котором время работы программы составляло 46,919 секунд.
2. Программа была скомпилирована компилятором GCC с уровнями оптимизации O0, -O1, -O2, -O3, -Os, -Ofast, -Ogпод архитектуру процессора x86 (x86-64). Для каждого из семи вариантов было измерено время работы программы для нескольких значений входного параметра N (0.25\*N0 = 25\*107, 0.5\*N0 = 5\*108, N0 = 109, 1.5\*N0 = 15\*108), для каждого значения было выполнено три независимых измерения времени.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0,25\*N0 (c)** | **с/a (с)** | **0,5\*N0 (с)** | **с/а (с)** | **N0 (с)** | **с/а (с)** | **1,5\*N0 (с)** | **с/а (с)** |
| **O0** | 11,722  11,736  11,743 | 11,736 | 23,454  23,430  23,522 | 23,469 | 46,836  46,820  46,825 | 46,827 | 70,384  70,233  70,226 | 70,281 |
| **O1** | 11,714  11,743  11,738 | 11,731 | 23,412  23,405  23,415 | 23,411 | 46,821  46,846  46,814 | 46,827 | 70,235  70,228  70,215 | 70,226 |
| **O2** | 11,718  11,717  11,713 | 11,716 | 23,421  23,416  23,424 | 23,420 | 46,815  46,822  46,820 | 46,819 | 70,233  70,231  70,220 | 70,228 |
| **O3** | 11,714  11,712  11,718 | 11,714 | 23,414  23,414  23,417 | 23,415 | 46,816  46,814  46,845 | 46,825 | 70,256  70,250  70,246 | 70,251 |
| **Os** | 11,714  11,706  11,716 | 11,712 | 23,404  23,411  23,408 | 23,408 | 46,818  46,819  46,812 | 46,816 | 70,227  70,221  70,230 | 70,226 |
| **Ofast** | 11,718  11,714  11,706 | 11,713 | 23,446  23,414  23,427 | 23,429 | 46,818  46,847  46,831 | 46,832 | 70,227  70,231  70,231 | 70,233 |
| **Og** | 11,713  11,710  11,711 | 11,711 | 23,420  23,418  23,418 | 23,419 | 46,820  46,811  46,822 | 46,818 | 70,226  70,239  70,237 | 70,234 |

с/а – среднее арифметическое трех независимых измерений, вычисляется для каждого уровня компиляции, для каждого значения входного параметра.

Зеленым цветом выделено наименьшее время для данного значения входного параметра.

1. По полученным средним арифметическим значениям был построен график.

ЗАКЛЮЧЕНИЕ

По результатам практической работы мы можем сделать вывод, что в случае, когда наша программа имеет малый размер (~50 строк) и реализует простой алгоритм (много раз не вызываются объемные функции), время выполнения программы отличается всего на сотые доли секунды в зависимости от выбранного уровня оптимизации. Это значит, что в случае, когда наша программа запускается небольшое количество (меньше 100) раз, разница во времени выполнения программы будет несущественна.

# Приложение 1. *Листинг реализованной программы*

#include <stdio.h>

#include <string.h>

double LeibnizFormula (long long N)

{

double Pi = 0, nextMember;

long long i;

for (i = 0; i < N; i++)

{

if (i % 2)

nextMember = -1;

else

nextMember = 1;

nextMember /= 2\*(double)(i) + 1;

Pi += nextMember;

}

Pi \*= 4;

return Pi;

}

int fromCharToInt (char c)

{

return (int)(c - '0');

}

long long getN (char \* charN)

{

int i, lengthN = strlen(charN);

long long N = 0;

for (i = 0; i < lengthN; i++)

{

N += (long long)(fromCharToInt(charN[i]));

N \*= 10;

}

return N;

}

void printAnswer(double Pi)

{

printf("%f\n",Pi);

}

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

{

long long N = getN(argv[1]);

double Pi = LeibnizFormula(N);

printAnswer(Pi);

return 0;

}

# Приложение 2. *Команды для компиляции и запуска программы*

Команды для компиляции программы:

gcc lab1.c -o lab1.out

gcc –O0 lab1.c –o lab1\_0.bin –Wall

gcc –O1 lab1.c –o lab1\_1.bin –Wall

gcc –O2 lab1.c –o lab1\_2.bin –Wall

gcc –O3 lab1.c –o lab1\_3.bin –Wall

gcc –Os lab1.c –o lab1\_s.bin –Wall

gcc –Ofast lab1.c –o lab1\_fast.bin –Wall

gcc –Og lab1.c –o lab1\_g.bin –Wall

Команды для запуска программы:

./lab1.out N

./lab1\_0.bin N

./lab1\_1.bin N

./lab1\_2.bin N

./lab1\_3.bin N

./lab1\_s.bin N

./lab1\_fast.bin N

./lab1\_g.bin N