###### 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

«ОПРЕДЕЛЕНИЕ ВРЕМЕНИ РАБОТЫ ПРИКЛАДНЫХ ПРОГРАММ»

и

«ИЗУЧЕНИЕ ОПТИМИЗИРУЮЩЕГО КОМПИЛЯТОРА»

студента 2 курса, группы 21209

**Усольцева Антона Андреевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

В.А.Перепелкин

Новосибирск 2022

**СОДЕРЖАНИЕ**

Цель 3

Задачи 3

Описание работы 4

Заключение 8

# ЦЕЛЬ

1. Изучение методик измерения времени работы программы;

2. Изучить основные функций оптимизирующего компилятора и

примеры оптимизирующих преобразований.

# ЗАДАЧИ

1. Написать программу на языке C или C++, которая решает некоторую тестовую задачу, и подобрать к ней входные параметры так, чтобы время ее работы составляло примерно 30 секунд;

2. Проверить правильность работы программы на нескольких тестовых

наборах входных данных;

3. Применить 3 способа измерения времени работы программы и ее частей – time(), clock\_gettime, times.

4. По приведенным методикам определить время работы подпрограммы

тестовой программы с относительной погрешностью не более 1%;

5. Программу скомпилировать компилятором GCC с уровнями оптимизации

-O0, -O1, -O2, -O3, -Os, -Ofast, -Og под архитектуру процессора x86, выполнить тестовые запуски, составить график зависимости времени исполнения от уровня оптимизации, проанализировать результаты.

# ОПИСАНИЕ РАБОТЫ

***ОПРЕДЕЛЕНИЕ ВРЕМЕНИ РАБОТЫ ПРИКЛАДНЫХ ПРОГРАММ***

Для данной лабораторной работы в качестве исполняющегося алгоритма будем использовать разложение синуса в ряд Тейлора :

Входные данные: два числа n и x

Листинг 1

#include "stdio.h"

#include "math.h"

#include "string.h"

int stoi(char \*str, unsigned int len) {

int res = 0;

for (int i = 0; i < len; i++) {

if (str[i] - '0' < 0 || str[i] - '0' > 9) return -1;

res = res \* 10 + str[i] - '0';

}

return res;}

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

int n = stoi(arcv[1], strlen(arcv[1]));

double x = stoi(arcv[2], strlen(arcv[2]));

double res = 0;

double fact = 1;

double a = 1;

for (int i = 1; i <= n; ++i) {

res += ((double) (a \* x) / fact);

a \*= -1;

x \*= pow(x, 2);

fact \*= ((2 \* i) \* (2 \* i + 1));

}

printf("%lf", res);

return 0;}

Способы измерения времени:

1. Утилита *time()*. Данная утилита является готовым решением, но из недостатков имеет то, что она замеряет время работы всей программы целиком, а также что она использует системное время, т.е. во время измерения учитываются другие процессы исполняемые процессором.  
   Скомпилируем код (листинг 1) командой   
    $ gcc timer1.c -o time1 –lm   
   в файл time1 и затем используем функцию time :  
    $ time ./time1 234858000 5   
   Переданные аргументы подобраны для того, чтобы программа выполнялась примерно 15 секунд. В результате исполнения функция time выводит три поля:   
   real 0m15.025s - общее время работы программы согласно системному таймеру.   
   user 0m15.000s - время, которое работал пользоватльеский процесс (кроме времени работы других процессов);  
   sys 0m0.031s- время, затраченное на выполнение системных вызовов программы.  
   Абсолютная точность такого измерения составляет 1мс.
2. Библиотечная функция *clock\_gettime*.   
   Также как и предыдущая функция, *clock\_gettime* использует разницу значений системного времени. Данная функция записывает данные времени в структуру из двух полей: tv\_sec и tv\_nsec , задающих количество секунд и наносекунд. Поэтому абсолютной точностью данной функции можно считать 1 нс.  
    Листинг 2  
   Скомпилируем код (листинг 2) командой   
    $ gcc timer2.c -o time2 -lrt -lm  
   и запустим с аргументами 234858000 5 . После исполнения получим время исполнения фрагмента кода: Time taken: 14.883099 sec. Заметим, что полученная точность составляет всего 1мкс(?).

#include "math.h"

#include "stdio.h"

#include "math.h"

#include "string.h"

#include <time.h> // for clock\_gettime

int stoi(char \*str, unsigned int len) {

int res = 0;

for (int i = 0; i < len; i++) {

if (str[i] - '0' < 0 || str[i] - '0' > 9) return -1;

res = res \* 10 + str[i] - '0';

}

return res;

}

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

struct timespec start, end;

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

int n = stoi(arcv[1], strlen(arcv[1]));

double x = stoi(arcv[2], strlen(arcv[2]));

double res = 0;

double fact = 1;

double a = 1;

for (int i = 1; i <= n; ++i) {

res += ((double) (a \* x) / fact);

a \*= -1;

x \*= pow(x, 2);

fact \*= ((2 \* i) \* (2 \* i + 1));

}

printf("%lf", res);

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;

}

1. Библиотечная функция *times*.  
   Данная функция, в отличие от предыдущих, измеряет время работы конкретного процесса.  
    Листинг 3  
   Скомпилируем код (листинг 3) командой   
    $ gcc timer3.c -o time3 -lrt -lm  
   и запустим с аргументами 234858000 5 . После исполнения получим время исполнения фрагмента кода: Time taken: 14.640000 sec. Точность такого измерения составляет 10мс (проверено многократными измерениями)

#include <stdio.h> // for printf

#include <sys/times.h> // for times

#include <unistd.h> // for sysconf

#include "math.h"

#include "string.h"

int stoi(char \*str, unsigned int len) {

int res = 0;

for (int i = 0; i < len; i++) {

if (str[i] - '0' < 0 || str[i] - '0' > 9) return -1;

res = res \* 10 + str[i] - '0';

}

return res;

}

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

struct tms start, end;

long clocks\_per\_sec = sysconf(\_SC\_CLK\_TCK);

long clocks;

times(&start);

int n = stoi(arcv[1], strlen(arcv[1]));

double x = stoi(arcv[2], strlen(arcv[2]));

double res = 0;

double fact = 1;

double a = 1;

for (int i = 1; i <= n; ++i) {

res += ((double) (a \* x) / fact);

a \*= -1;

x \*= pow(x, 2);

fact \*= ((2 \* i) \* (2 \* i + 1));

}

printf("%lf", res);

times(&end);

clocks = end.tms\_utime - start.tms\_utime;

printf("Time taken: %lf sec.\n", ((double) clocks / clocks\_per\_sec));

return 0;

}

**ВЫВОД**:

В ходе работы был написана программы, которая раскладывает функцию синуса в ряд Тейлора и затем для этого алгоритма были применены три различных способа измерения времени работы. Было установлено, что разные таймеры имеют разную абсолютную точность. Также в ходе проверки всех таймеров на одинаковом наборе данных были замечены частичные несоответствия времени исполнения. Это связано с принципом работы выбранных таймеров, а также с изменяющимися условиями системы (температура, фоновые процессы, заряд аккумулятора и тд).

***ИЗУЧЕНИЕ ОПТИМИЗИРУЮЩЕГО КОМПИЛЯТОРА***

Возьмем код из Листинга 1 для исследования влияния оптимизационных настроек компилятора GCC на время исполнения программы.

Проведем по три измерения с различными входными данными для каждого уровня оптимизации и занесем данные в таблицу.

Измерять время будем с помощью утилиты time().

Время в таблице выражается в секундах.

Таблица 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | -O0 | -O1 | -O2 | -O3 | -Os | -Ofast | -Og |
| 1000000 | 0.020 | 0.013 | 0.015 | 0.016 | 0.019 | 0.014 | 0.014 |
| 1000000000 | 4.612 | 2.223 | 2.168 | 2.263 | 7.778 | 1.681 | 2.265 |
| 2000000000 | 9.277 | 4.513 | 4.399 | 4.526 | 15.419 | 3.339 | 4.534 |

**ВЫВОД**:

В ходе работы были применены различные уровни оптимизации при компилировании программы, и затем было измерено время исполнения программы. Для каждого из трех входных значений зависимость времени исполнения от уровня оптимизации оказалась примерно одинакова: t(O0)>t(O1);

t(O1) ≈ t(O2) ≈ t(O3) ≈ t(Og);

t(O3)< t(O0)<t(Os);

t(Ofast) – наименьшее, t(Os) – наибольшее.

где t(Ox) – время исполнения программы при уровне оптимизации х.

Самым оптимизирующим оказался уровень оптимизации Ofast. Это связано с тем, что Ofast включает все оптимизации уровня O3, а также ряд других, таких как использование более быстрых и менее точных математических функций, а исследуемая программа почти полностью состоит из математических функций. Также хоть и t(O1) ≈ t(O2) ≈ t(O3) ≈ t(Og) но

t(O3) немного больше чем t(O2). Это связано с тем, что в уровне O3 включены все оптимизации из уровня O2, но к ним добавлены оптимизации времени работы программы, которые могут приводить к увеличению размера бинарного исполняемого файла и большим затратам в памяти.

Уровень оптимизации Os оказался наименее эффективным.

**ЗАКЛЮЧЕНИЕ**

Были изучены различные способы измерения времени исполнения программы или ее подпрограммы. С помощью них было получено время исполнения программы разложения синуса в ряд Тейлора. Также были изучены различные уровни оптимизации компиляции и измерено их влияние на время исполнения программы. Поставленные цели и задачи были выполнены.