Новосибирский государственный университет

Лабораторная работа №1-2

Выполнил: Леонтьев Данил Вячеславович

2022

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

Вариант задания:

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

Описание методики для определения времени работы программы и результат

В данной лабораторной работе использовалась утилита, работающая по системному таймеру, time

Результат: 14,052 секунды

Полный компилируемый листинг реализованной программы и команда для ее компиляции

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <math.h>

#include <stdio.h>

#include <time.h>

int main() {

struct timespec start, end;

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

long int N;

double pi = 0,b;

scanf("%ld", &N);

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

pi=-pi;

b=4/(double)(2\*i+1);

pi=pi+b;

}

if (pi<0)

pi=-pi;

printf("%lf\n", pi);

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;

}

gcc lab1.c –o lab1.bin

Графики работ программ с оптимизациями от параметра N

Полный компилируемый листинг реализованной программы и команды для ее компиляции

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <math.h>

#include <stdio.h>

#include <time.h>

int main() {

long long N;

double pi = 0,b;

scanf("%lld", &N);

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

pi=-pi;

b=4/(double)(2\*i+1);

pi=pi+b;

}

if (pi<0)

pi=-pi;

printf("%lf\n", pi);

return 0;

}

gcc –O0 lab2.c –o lab2.bin –Wall

gcc –O1 lab2.c –o lab2.bin –Wall

gcc –O2 lab2.c –o lab2.bin –Wall

gcc –O3 lab2.c –o lab2.bin –Wall

gcc –Os lab2.c –o lab2.bin –Wall

gcc –Ofast lab2.c –o lab2.bin –Wall

gcc –Og lab2.c –o lab2.bin –Wall

Вывод

В результате двух проделанных лабораторных работ я обучился метода фиксирования времени проведения программы и ускорением программ путем оптимизации компилятора. Так же эмпирическим путем я выявил, что оптимизация компилятора практически не влияет на элементарные программы, но при этом и есть виды оптимизаций, которые наоборот значительно увеличивают время выполнения программы.