МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа №1**

«Исследование алгоритмических и программных методов ускорения реализации функций вещественных переменных»

Вариант №3

Выполнил:

студент 3 курса, гр. ИВТВМбд-31

Захарычев Н.А.

Проверил:

д. т. н., профессор кафедры ВТ

Негода В. Н.

г. Ульяновск, 2017

**Цель работы:** Изучение методов реализации функций от вещественных переменных, представленных степенными рядами. Приобретение умений и навыков варьирования соотношения «затраты памяти – время реализации» в рамках этих методов.

f(x) = точность – 22 чисел после двоичной точки.

**1. Анализ разложения в ряды функций, фигурирующих в варианте задания.**

Разложение косинуса в степенной ряд:

Область сходимости:

Преобразование ряда по схеме Горнера для 5 членов ряда:

**2. Разработка процедуры-функции контроля**

**Определение количества членов ряда**

Чтобы найти длину ряда, нужно написать функцию, которая будет сравнивать результат нашей функции с эталоном – *cos(x)*

Погрешность вычислений не должна превышать:



Фрагмент кода определяющий достаточное количество членов ряда:

for (double i = 0.0; i < 1; i += max\_delta){

while (flverify\_cos(i, fl\_cycle\_nogorner) == 0){

n++;

}

}

// функция для сравнения с эталоном при прототипировании

int flverify\_cos(float fl, PFLOAT p) {

float cosstd = cos(fl);

float mycos = p(fl);

float delta = fabs(cosstd - mycos);

if (delta < max\_delta)

return 1;

return 0;

}

В результате прототипирования выяснилось, что необходимая точность достигается при 5 членах ряда.

**3. Исследование времени вычисления для данных типа float**

Эталонная функция:

float FlMath(float x) {

float res = cos(x);

return res;

}

Наивная реализация с циклом без схемы Горнера:

float fl\_cycle\_nogorner(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorial(2\*i)));

}

return result;

}

Наивная реализация без цикла без схемы Горнера:

float fl\_nocycle\_nogorner(float x) {

return pow(-1, 0) \* (pow(x, 2 \* 0) / (factorial(2 \* 0))) +

pow(-1, 1) \* (pow(x, 2 \* 1) / (factorial(2 \* 1))) +

pow(-1, 2) \* (pow(x, 2 \* 2) / (factorial(2 \* 2))) +

pow(-1, 3) \* (pow(x, 2 \* 3) / (factorial(2 \* 3))) +

pow(-1, 4) \* (pow(x, 2 \* 4) / (factorial(2 \* 4))) +

pow(-1, 5) \* (pow(x, 2 \* 5) / (factorial(2 \* 5)));

}

Реализация без цикла со схемой Горнера:

float fl\_nocycle\_gorner(float x) {

float double\_x = x \* x;

return 1 -

double\_x \* (((1 / (factorial(2))) -

double\_x \* ((1 / (factorial(4))) -

double\_x \* ((1 / (factorial(6))) -

double\_x \* ((1 / (factorial(8))) -

double\_x \* ((1 / (factorial(10)))))))));

}

Реализация с циклом со схемой Горнера:

float fl\_cycle\_gorner(float x) {

float res = 0;

res = (double)1.0 / factorial(8) - x\*x / factorial(10);

for (int i = 3; i >= 0; i--) {

res \*= x\*x;

res = (double)1.0 / (factorial(2 \* i)) - res;

}

return res;

}

**Оценка быстродействия**

Все замеры производились с помощью функции *clock().* Вычислялось среднее значение на 10 000 000 проходов, при одинаковом значении аргумента.

Функция оценки быстродействия:

int getTackt(int count, PFLOAT p) {

double duration;

clock\_t start = clock();

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

p(1);

}

double end = (clock() - start);

duration = end \* pow(10.0, 9) / (double)CLOCKS\_PER\_SEC / count;

return (int)duration;

}

|  |  |
| --- | --- |
| **Функция** | **Время работы, нс** |
| FlCyclNoGorner | 1070 |
| FlNoCyclNoGorner | 983 |
| FlNoCycleGorner | 151 |
| FlCyclGorner | 244 |
| FlMath | 94 |

Можно заметить, что любая бесцикловая реализация обгоняет реализацию с циклом, это обусловлено тем, что в цикловой есть операции для организации цикла, которые отсутствуют в бесцикловой.

**4. Разработка макросов обработки чисел с фиксированной точкой.**

Существует проблема, когда производительность приложения может заметно ухудшиться из-за особенностей вычисления на числах с плавающей точкой. Данной проблеме существует решение, которое позволяет проводить вычисления с фиксированной точкой на целочисленном типе.

Перед тем, как начать измерения времени выполнения процедур с фиксированной точкой, было принято решение использовать 32-битный тип - \_\_int32. Для того чтобы обеспечить достаточную точность, под дробную часть был отведен 21 бит, следовательно:

11 бит – целая часть, 21 бит – дробная.

Определение типа с фиксированной точкой:

typedef \_\_int32 PFIX; // определение типа с фиксированной точкой

Преобразование из плавающей точки в фиксированную:

#define fix11\_21(a) ((PFIX)((a) \* (1LL<<21)))

Преобразование из фиксированной точки в плавающую:

#define fix\_to\_float(a) ((a) / (double)(1LL<<21))

Операция умножения с числами с фиксированной точкой:

#define fix\_mul(a, b) ((PFIX)(((\_\_int64)(a) \* (b)) >> 21))

Операция деления с числами с фиксированной точкой:

#define fix\_div(a, b) ((PFIX)(((\_\_int64)(a) << 21) / (b)))

Указатель на функцию с фиксированной точкой:

typedef PFIX(\*FUNC\_FIX)(PFIX); // Указатель на функцию фикс. точка

**5. Исследование времени вычисления для данных с фиксированной точкой**.

Реализация с фиксированной точкой с циклом со схемой Горнера:

PFIX fix\_cycle\_gorner(PFIX x) {

PFIX res = fix1\_31((double)1.0 / factorial(8) - x\*x / factorial(10));

for (int i = n; i >= 0; i--) {

res = fix\_mul(x, x);

res = fix\_div(1, fix1\_31(factorial(2 \* i))) - res;

}

return res;

}

Реализация с фиксированной точкой без цикла со схемой Горнера:

PFIX fix\_nocycle\_gorner(PFIX x) {

PFIX double\_x = fix\_mul(x, x);

PFIX res = 1 -

fix\_mul(double\_x, fix\_div(1, fix1\_31(factorial(2))) -

fix\_mul(double\_x, fix\_div(1, fix1\_31(factorial(4))) -

fix\_mul(double\_x, fix\_div(1, fix1\_31(factorial(6))) -

fix\_mul(double\_x, fix\_div(1, fix1\_31(factorial(8))) -

fix\_mul(double\_x, fix\_div(1, fix1\_31(factorial(10))))))));

return res;

}

**Оценка быстродействия**

Все замеры производились с помощью функции *clock().* Вычислялось среднее значение на 10 000 000 проходов, при одинаковом значении аргумента.

Функция оценки быстродействия:

int getTackt\_fix(int count, FUNC\_FIX p) {

double duration;

clock\_t start = clock();

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

p(1);

}

double end = (clock() - start);

duration = end \* pow(10.0, 9) / (double)CLOCKS\_PER\_SEC / count;

return (int)duration;

}

|  |  |
| --- | --- |
| **Функция** | **Время работы, нс** |
| FlNoCycleGorner | 151 |
| FlCyclGorner | 244 |
| FixCycleGorner | 306 |
| FixNoCycleGorner | 273 |
| FlMath | 94 |

Используя числа с фиксированной точкой, мы не получили выигрыша по времени, поскольку при умножении и делении использовалось приведение типов, следовательно, мы имеем большее количество машинных команд относительно умножения и деления для плавающей точки, что существенно замедляет алгоритм, также значения получились такими большими, исходя из того, что для фиксированной точки использовался тип в два раза больше чем для плавающей точки.

**6. Исследование таблично-алгоритмических реализаций функций.**

Попробуем получить выигрыш по скорости реализуя табличный метод с плавающей точки.

void prepare\_table() {

for (int i = 0; i < table\_count; i++){

table[i] = new float[3];

float x = (float)i / table\_count;

table[i][0] = cos(x);

table[i][1] = (-1) \* sin(x);

table[i][2] = (-1) \* cos(x);

}

int test1=4;

return;

}

float table\_method(float x) {

int a = x \* k;// старшие k бит

float h = (x \* k - a) / table\_count;

float b0 = table[a][0];

float b1 = table[a][1];

// float b2 = table[a][2];

float res = b0 + b1\*h;

// float res = b0 + b1\*h + b2\*h\*h;

return res;

}

Для необходимой точности достаточно 8 старших разрядов, при 2-x членах ряда.

|  |  |
| --- | --- |
| **Функция** | **Время работы, нс** |
| FlNoCycleGorner | 151 |
| FlTable | 32 |

Таблично-алгоритмический метод для плавающей запятой выигрывает по скорости бесцикловую реализацию схемы Горнера.

**7. Формирование итоговых результатов**

|  |  |
| --- | --- |
| **Функция** | **Время работы, нс** |
| FlCyclNoGorner | 1070 |
| FlNoCyclNoGorner | 983 |
| FlNoCycleGorner | 151 |
| FlCyclGorner | 244 |
| FlMath | 94 |
| FixNoCycleGorner | 273 |
| FixCyclGorner | 306 |
| FlTable | 32 |

**Вывод:** из всех реализованных алгоритмов большинство выигрывают по скорости у стандартного cos(x) из библиотеки math.h, наибольший выигрыш мы получаем в таблично-алгоритмическом методе для плавающей точки.

**Исходный код**

#include "stdafx.h"

#include <stdio.h>

#include <math.h>

#include <conio.h>

#include <time.h>

#include <iostream>

using namespace std;

typedef float(\*PFLOAT)(float);

//float calcLn(float x);

float max\_delta = (float)pow(2, -22);

int n = 5;

float factorials[100];

typedef \_\_int32 PFIX;

#define fix1\_31(a) ((PFIX)((a) \* (1LL<<7)))

#define fix\_to\_float(a) ((a) / (double)(1LL<<7))

#define fix\_mul(a, b) ((PFIX)(((\_\_int64)(a) \* (b)) >> 7))

#define fix\_div(a, b) ((PFIX)(((\_\_int64)(a) << 7) / (b)))

typedef PFIX(\*FUNC\_FIX)(PFIX); // Указатель на функцию фикс. точка

int fixverify(float fl, FUNC\_FIX p); // указатель на функцию

bool test\_fix(FUNC\_FIX p); // проверка на достаточное кол-во членов ряда фикс. точка

int getTackt\_fix(int count, FUNC\_FIX p); // получить кол-во тактов фикс. точка

PFIX FixCycleGorner(PFIX x); // реализация со схемой горнера с циклом фикс. точка

PFIX FixNoCycleGorner(PFIX x); // реализация со схемой горнера без цикла фикс. точка

typedef float(\*PFLOAT)(float); // Указатель на функцию

//float max\_delta = (float)pow(2.0, -22); // макс погрешность

int flverify(float fl, PFLOAT p); // указатель на функцию

float calcCos(float x); // косинус свой

double fact(double x); // факториал

float fl\_cycl\_noGorner(float x); // простая реализация с циклом без горнера

float fl\_nocycl\_nogorner(float x); // простая реализация без цикла без горнера

float fl\_nocycle\_gorner(float x); // реализация со схемой горнера без цикла

float fl\_cycl\_gorner(float x); // реализация со схемой горнера с циклом

float FlMath(float x); // стандартная реализация

int getTackt(int count, PFLOAT p); // получить кол-во тактов

bool test(PFLOAT p); // проверка на достаточное кол-во членов ряда

void prepare\_table(); // генерация таблицы

float table\_method(float x); // табличный метод

float tableMetFl(float arg);

double myPow(float x, int n);

double factorial(int x) {

if (factorials[x] == 0) {

factorials[x] = (x == 0 ? 1 : factorial(x - 1) \* x);

}

return factorials[x];

}

float calcCos(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorial(2 \* i)));

}

return result;

}

int getTackt(int count, PFLOAT p) {

double duration;

clock\_t time = clock();

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

p(1);

}

double end = (clock() - time);

duration = end \* pow(10.0, 9) / (double)CLOCKS\_PER\_SEC / count;

return (int)duration;

}

float fl\_cycle\_nogorner(float x) {

float result = 0;

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

result += pow(-1, i) \* (pow(x, 2 \* i) / (factorial(2 \* i)));

}

return result;

}

float fl\_cycle\_gorner(float x) {

float res = 0;

res = (double)1.0 / factorial(8) - x\*x / factorial(10);

for (int i = 3; i >= 0; i--) {

res \*= x\*x;

res = (double)1.0 / (factorial(2 \* i)) - res;

}

return res;

}

PFIX fix\_cycle\_gorner(PFIX x) {

PFIX pow = fix\_mul(x, x);

PFIX d = fix1\_31(1);

PFIX res = fix\_div(d, fix1\_31(factorial(8))) - fix\_div(pow, fix1\_31(factorial(10)));

for (int i = 3; i >= 0; i--) {

res = fix\_mul(pow, res);

res = fix\_div(d, fix1\_31(factorial(2 \* i))) - res;

}

return res;

}

int flverify\_cos(float fl, PFLOAT p) {

float cosstd = cos(fl);

float mycos = p(fl);

float delta = fabs(cosstd - mycos);

if (delta < max\_delta)

return 1;

return 0;

}

bool test(PFLOAT p) {

for (float i = max\_delta; i < 1; i += max\_delta) {

if (flverify\_cos(i, p) == 0)

return false;

}

return true;

}

float FlMath(float x) {

float res = cos(x);

return res;

}

bool test\_fix(FUNC\_FIX p) {

for (float i = max\_delta; i < 1; i += max\_delta) {

if (fixverify(i, p) == 0)

return false;

}

return true;

}

int fixverify(float x, FUNC\_FIX p) {

float cosfix = cos(x);

PFIX myfixcos = p(fix1\_31(x));

float res2 = fix\_to\_float(myfixcos);

if (fabs(cosfix - res2) > max\_delta)

return 0;

return 1;

}

//циклическая схема горнера

float fl\_nocycle\_gorner(float x) {

float double\_x = x \* x;

return 1 -

double\_x \* (((1 / (factorial(2))) -

double\_x \* ((1 / (factorial(4))) -

double\_x \* ((1 / (factorial(6))) -

double\_x \* ((1 / (factorial(8))) -

double\_x \* ((1 / (factorial(10)))))))));

}

PFIX fix\_nocycle\_gorner(PFIX x) {

PFIX double\_x = fix\_mul(x, x);

PFIX d = fix1\_31(1);

PFIX res = d -

fix\_mul(double\_x, fix\_div(d, fix1\_31(factorial(2))) -

fix\_mul(double\_x, fix\_div(d, fix1\_31(factorial(4))) -

fix\_mul(double\_x, fix\_div(d, fix1\_31(factorial(6))) -

fix\_mul(double\_x, fix\_div(d, fix1\_31(factorial(8))) -

fix\_mul(double\_x, fix\_div(d, fix1\_31(factorial(10))))))));

return res;

}

float fl\_nocycle\_nogorner(float x) {

return pow(-1, 0) \* (pow(x, 2 \* 0) / (factorial(2 \* 0))) +

pow(-1, 1) \* (pow(x, 2 \* 1) / (factorial(2 \* 1))) +

pow(-1, 2) \* (pow(x, 2 \* 2) / (factorial(2 \* 2))) +

pow(-1, 3) \* (pow(x, 2 \* 3) / (factorial(2 \* 3))) +

pow(-1, 4) \* (pow(x, 2 \* 4) / (factorial(2 \* 4))) +

pow(-1, 5) \* (pow(x, 2 \* 5) / (factorial(2 \* 5)));

}

double myPow(float x, int n) {

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

x = x \* x;

}

return x;

}

int getTackt\_fix(int count, FUNC\_FIX p) {

double duration;

clock\_t start = clock();

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

p(1);

}

double end = (clock() - start);

duration = end \* pow(10.0, 9) / (double)CLOCKS\_PER\_SEC / count;

return (int)duration;

}

const int k = 8; //бит адреса 8

int table\_count = 1 << k; // размер таблицы

float \*\*table = new float\*[table\_count];

void prepare\_table() {

for (int i = 0; i < table\_count; i++) {

table[i] = new float[3];

float x = (float)i / table\_count;

table[i][0] = cos(x);

table[i][1] = (-1) \* sin(x);

table[i][2] = (-1) \* cos(x);

}

int test1 = 4;

return;

}

float table\_method(float x) {

int a = x \* k;// старшие k бит

float h = (x \* k - a) / table\_count;

float b0 = table[a][0];

float b1 = table[a][1];

float res = b0 + b1\*h;

return res;

}

float tableMetFl(float arg) {

int index = int(arg\*(1 << k));

float x = arg - float(index / (1 << k));

float result = table[index][0] + table[index][1] \* x + table[index][2] \* x \* x;

return result;

}

int main()

{

for (int i = 0; i < 10; i++) {

factorial(i);

}

printf("FlCyclNoGorner: %d\n", getTackt(1000000, fl\_cycle\_nogorner));

printf("FlNoCyclNoGorner: %d\n", getTackt(1000000, fl\_nocycle\_nogorner));

printf("FlNoCycleGorner: %d\n", getTackt(1000000, fl\_nocycle\_gorner));

printf("FlCyclGorner: %d\n", getTackt(1000000, fl\_cycle\_gorner));

printf("Std: %d\n", getTackt(1000000, FlMath));

printf("FixNoCycleGorner: %d\n", getTackt\_fix(1000000, fix\_nocycle\_gorner));

printf("FixCycleGorner: %d\n", getTackt\_fix(1000000, fix\_cycle\_gorner));

prepare\_table();

printf("TableMtd: %d\n", getTackt(1000000, table\_method));

\_getch();

return 0;}