Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

09.03.01 Информатика и вычислительная техника

Профиль: Программное обеспечение средств вычислительной техники и автоматизированных систем

**Отчёт по лабораторной работе №3**

**по дисциплине «Архитектура вычислительных систем»**

по направлению 09.03.01 «Информатика и вычислительная техника»,

направленность (профиль) – «Программное обеспечение средств вычислительной техники и автоматизированных систем», квалификация – бакалавр,

программа академического бакалавриата,

форма обучения – очная, год начала подготовки (по учебному плану) – 2016

Выполнил: студент гр. ИП-611 / Макаревич А.А. /

Проверил: преподаватель кафедры ВС / Токмашева Е.И. /

Новосибирск, 2018

**Введение**

**Цель лабораторной работы:** измерить время выполнения исходного кода, и определить метод, который точнее сможет посчитать время выполнения моей программы.

**Задачи:** написать программу на языке С++ реализующую численное интегрирование по формуле Симпсона с возможностью изменения необходимых входных данных. Оценить в секундах время выполнения выбранной математической функции. Оценить точность и погрешность (абсолютную и относительную) измерения времени. Для каждого способа измерения построить график зависимости времени выполнения функции от количества запусков. Составить отчёт с результатами выполнения работы.

**Предмет исследования:** программа, реализующая численное интегрирование по формуле Симпсона.

**Средства,** используемые при проведении исследования: интегрированная среда разработки программного обеспечения Microsoft Visual Studio 2012, Windows API.

**Выполнение работы**

Для оценки времени выполнения моего исходного кода(численное интегрирование по формуле Симпсона), который содержится в исходном файле **avs.cpp**, я использовали библиотеку cmath для функции **clock()**, и библиотеку windows.h для функций **GetTickCount()**, **QueryPerformanceCounter()** и **QueryPerformanceFrequency()** и регистра счётчика **TSC**.

В соответствии со стандартом ANSI C в библиотеке функций должен существовать файл сtime, в котором определены функции для работы со временем. Таким образом, функции, определенные в ANSI C не зависят от операционной системы и аппаратной платформы.

#include <ctime>

clock\_t clock(void);

Функция **clock()** возвращает число тактов таймера с момента запуска программы. Для преобразования результата в секунды его нужно разделить на макроопределение **CLOCKS\_PER\_SEC**. Выдается общее количество процессорного времени, прошедшего с момента начала выполнения программы в единицах, определенных машинно-зависимым макро **CLOCKS\_PER\_SEC**. Если такое измерение провести нельзя, то выдается -1. В большинстве реализаций библиотек C функция **clock()** имеет разрешение в 1 миллисекунду.

Функции Windows целесообразно использовать в тех случаях, когда код программы не планируется переносить на другие платформы.

#include <windows.h>

DWORD GetTickCount();

Функция **GetTickCount()** возвращает время с момента загрузки Windows в миллисекундах. Максимальное значение - 49.7 дней, после чего счет начинается сначала. В большинстве случаев предпочтительно использование именно этой функции Windows.

BOOL QueryPerformanceCounter( LARGE\_INTEGER \*lpPerformanceCount );

Функция **QueryPerformanceCounter()** возвращает значение таймера высокой точности. Позиционируется как таймер высокой точности, однако реальная точность возвращаемого значения зависит от версии Windows. Особенность использования данной функции заключается в том, что счетчик связан с конкретным процессором, что требует, чтобы текущий поток выполнялся на одном и том же процессоре (или его ядре). Для этого следует устанавливать маску допустимых процессоров функцией SetThreadAffinityMask.

BOOL QueryPerformanceFrequency( LARGE\_INTEGER \*lpFrequency );

Функция **QueryPerformanceFrequency()** возвращает частоту таймера высокой точности. Если возвращаемое значение 0 – таймер не существует или получить значение частоты не удалось. В документации MSDN не упоминается, что возвращаемое значение является частотой процессора, однако приближенно можно считать возвращаемую величину именно частотой процессора, а значение, возвращаемое **QueryPerformanceCounter()** значением счетчика тактов ядра процессора (у каждого ядра свой счетчик тактов).

В итоге мы имеем 4 способа вычисления времени работы исходного кода.

- clock();

- GetTickCount();

- QueryPerformanceCounter().

- регистр счётчика TSC.

Еще одним способом измерения времени выполнения фрагмента программы является использование WinAPI-функций **QueryPerformanceCounter()** и **QueryPerformanceFrequency().**

Функция **QueryPerformanceCounter()** получает текущее значение высокоточного счетчика, функция **QueryPerformanceFrequency()** возвращает число отсчетов выбранного счетчика в секунду.

Другим способом измерения времени является использовани есчетчика **TSC (Time Stamp Counter)**, представляющего собой 64-битный MSR-регистр (Model Specific Register) в составе современных процессоров. В момент включения компьютера его значение обнуляется, по приходу каждого нового импульса от тактового генератора значение счетчика инкрементируется. Большой диапазон значений счетчика гарантирует отсутствие его переполнения за время работы компьютера (для процессора, работающего на частоте 4 ГГц, время, необходимое для переполнения счетчика, составляет приблизительно 136 лет). **Счетчик TSC** может быть использован для измерения времени выполнения заданного фрагмента программы с высокой точностью (до 50–100 тактов или 10–20 нс). Для чтения значения счетчика существует команда **RDTSC**, позволяющая получение значения счетчика в паре регистров EDX:EAX. Данная команда не является привилегированной и допускает чтение счетчика из программы, запущенной на третьем кольце защиты (ring 3). Для чтения значений большинства других MSR-регистров необходимо находиться на нулевом кольце защиты (ring 0), что невозможно из обычной программы и требует разработки специального драйвера.

Для компиляции своей программы я использовал командную строку разработчика Microsoft Visual Studio 2012, где ввёл соответствующую команду.

> cl /EHsc avs.cpp

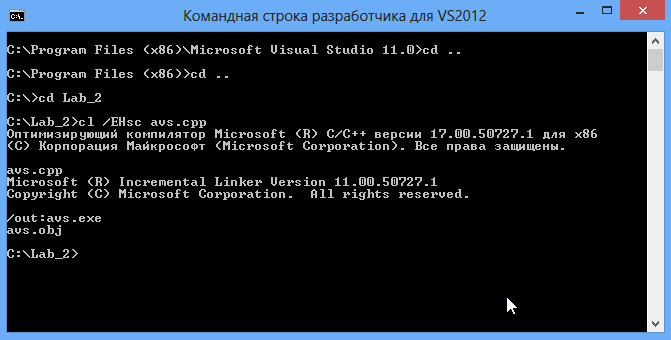


Рисунок 1. Компиляция программы.

> avs.exe

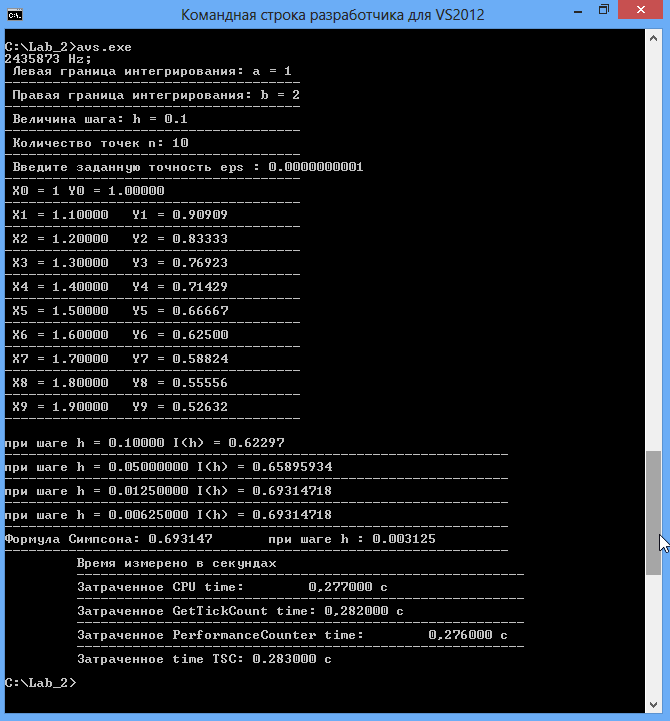


Рисунок 2. Результат запуска программы.

Мы получили время выполнения, выраженное в секундах, исходного кода четырьмя разными способами.

Независимо от используемого таймера, измерение времени работы программы всегда выполняется с некоторой погрешностью (абсолютной или относительной). Погрешность измерения используемого таймера не должна выходить за рамки допустимой точности измерения. Величина допустимого значения погрешности определяет тип таймера, которым следует пользоваться при определении времени работы программы. **Абсолютная погрешность** определяется разницей, например, в секундах между временем таймера и точным временем выполнения программы. Поскольку точное время выполнения программы никогда не известно, то абсолютная погрешность измерения оценивается точностью измерительного прибора. Например, если точность таймера 1 мс, то время было измерено с погрешностью не более 1 мс.

**Относительная погрешность** – это отношение абсолютной погрешности к величине временного интервала. Например, абсолютная погрешность в 1 мс даст относительную погрешность в 50%, если весь интервал был 2 мс и 0,1%, если интервал был 1 с. Таким образом, относительная погрешность показывает величину погрешности относительно всего интервала времени.

Точность функции **clock()** – 20 мс, тогда абсолютная погрешность измерения будет не более 20 мс. Если весь интервал был 277 мс, то он даст относительную погрешность 0.0722%.

Точность функции **GetTickTime()** – 16 мс, тогда абсолютная погрешность измерения будет не более 16 мс. Если весь интервал был 282 мс, то он даст относительную погрешность 0.0567%.

Точность функции **QueryPerformanceCounter()** - <1 мкс, абсолютная погрешность измерения будет не более 1 мкс. Весь интервал был 276 мс, то он даст относительную погрешность 0.00000362%.

Точность **регистра счётчика TSC** – 10 нс, абсолютная погрешность измерения будет не более 10 нс. Весь интервал был 283 мс, то он даст относительную погрешность 0.0000000353%.

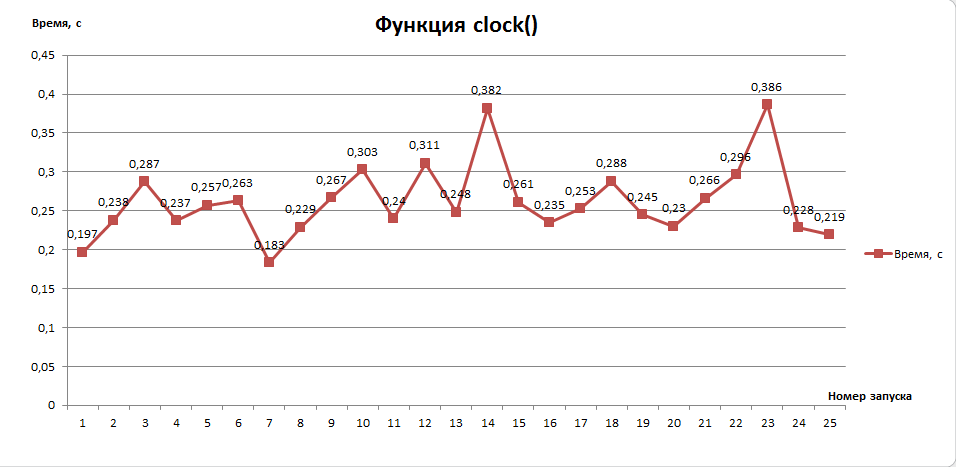
**Описание полученных результатов**

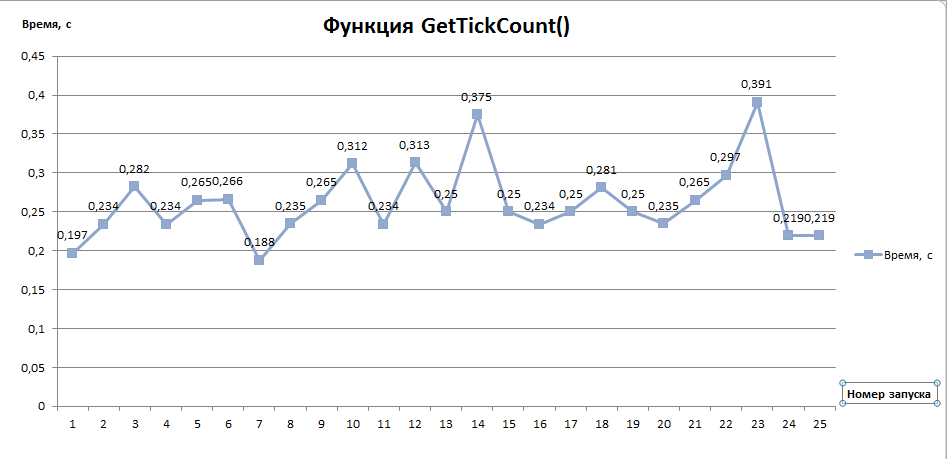
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Замер | 1 | 2 | 3 | 4 | 5 | 6 |
| clock() | 0.309 c | 0.238 c | 0.231 c | 0.313 c | 0.179 c | 0.238 c |
| GetTickCount() | 0.312 c | 0.234 c | 0.234 c | 0.313 c | 0.188 c | 0.250 c |
| QueryPerformanceCounter() | 0.309 c | 0.237 c | 0.231 c | 0.313 c | 0.179 c | 0.236 c |
| Регистр счётчика TSC | 0.316 c | 0.243 c | 0.236 c | 0.320 c | 0.183 c | 0.243 c |

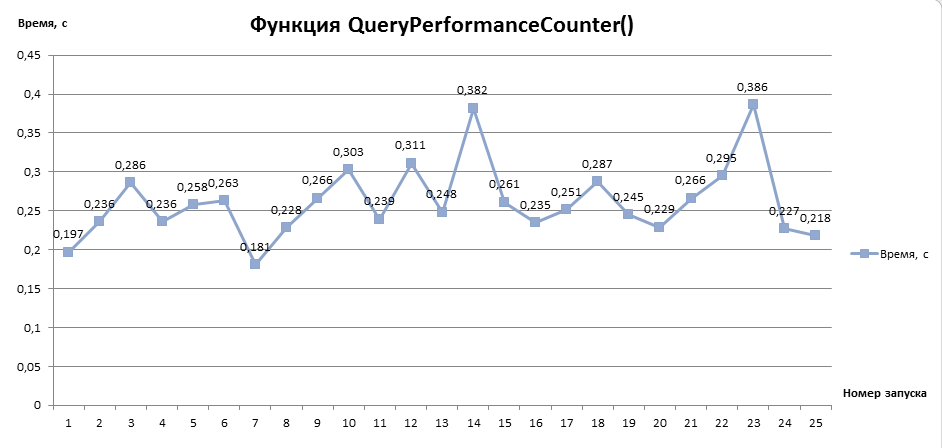
Из результатов измерений написанной мной программы, которая была на несколько раз запущена, можно сделать вывод, что время выполнения фрагмента исходного кода различно.

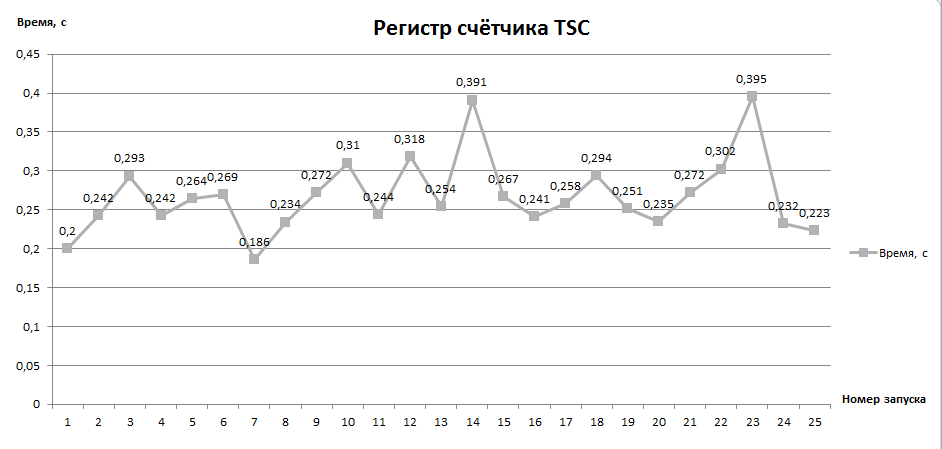
**Графики зависимости времени выполнения от количества запусков**

Построение графиков я выполнял в программе Microsoft Excel, с помощью созданных мной таблиц на основе результатов моей программы.









Исходя из графика каждого таймера, видно, что результат измерения одного и того же фрагмента кода выдаёт различные значения в силу того, что процессор и система могут исполнять и другие процессы параллельно нашей программы.

**Выводы по проделанной работе:**

В результате выполнения лабораторной работы, мы приобрели навыки вычисления времени какого-либо фрагмента исходного кода, а именно в моём случае, численное интегрирование по формуле Симпсона, смогли высчитать затраченное время на выполнение программы четырьмя способами, включая такие как: функции clock() библиотеки <ctime>, функции GetTickCount() и QueryPerformanceCounter() библиотеки <windows.h>, а также регистр счётчика TSC, и теперь, после своих вычислений, мы располагаем информацией о точности и погрешности каждого из способов. Для измерения времени выполнения исходного кода я бы выбрал два последних способа – это QueryPerformanceCounter() и регистр счётчика TSC, которые дают более высокую точность и надёжность вычисления времени, и имеющие низкие погрешности относительно всех остальных.

**Листинг программы**

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <iomanip>

#include <iostream>

#include <math.h>

#include <ctime>

using namespace std;

double Function(double x) {

return 1/x;

}

void Simpson(double a, double b, double h, double eps) {

int counter = 0;

double i;

double value = 0, value1 = 0;

do {

value1 = value;

value = Function(a) + Function(b);

for (i = a + h; i < b - h; i = i + 2\*h) {

value = value + 4\*Function(i);

}

for (i = a + h\*2; i <= b - h; i = i + 2\*h) {

value = value + 2\*Function(i);

}

value = h/3 \* value;

cout << "при шаге h = " << h << " I(h) = " << value << fixed << setprecision(8) << endl;

cout << "---------------------------------------------------------------" << endl;

h = h/2;

counter++;

}

while (counter < 2);

if (fabs(value - value1) <= 16\*eps) {

cout << "Формула Симпсона: " << fixed << setprecision(6) << value << "\t" << " при шаге h : " << h << endl;

cout << "---------------------------------------------------------------" << endl;

}

else Simpson(a, b, h/2, eps);

}

int main()

{

clock\_t c0, c1;

DWORD tt0, tt1;

// Средства PerformanceCounter

LARGE\_INTEGER pct0, pct1, freq;

QueryPerformanceFrequency(&freq); // Определяем частоту таймера

HANDLE hThread = GetCurrentThread(); // Устанавливаем выполнение текущего потока на первом ядре процессора

DWORD\_PTR prevMask = SetThreadAffinityMask(hThread, 1 ); // Завершаем квант процессорного времени для гарантированного переключения на задний процессор

Sleep(0);

long long frequency;

if (QueryPerformanceFrequency((LARGE\_INTEGER \*)&frequency))

cout << frequency << " Hz; " << endl;

else cout << "FAIL" << endl;

setlocale(LC\_ALL, "Russian");

double h, value = 0, value1 = 0, eps;

int i, numberpoints;

double a, b;

cout << " Левая граница интегрирования: a = ";

cin >> a;

cout << "-------------------------------------" << endl;

cout << " Правая граница интегрирования: b = ";

cin >> b;

cout << "-------------------------------------" << endl;

double \*f;

cout << " Величина шага: h = ";

cin >> h;

cout << "-------------------------------------" << endl;

cout << " Количество точек n: ";

cin >> numberpoints;

cout << "-------------------------------------" << endl;

cout << " Введите заданную точность eps : ";

cin >> eps;

cout << "-------------------------------------" << endl;

// Измеряем время начала отсчёта 4 способами

c0 = clock(); // Способ 1

tt0 = GetTickCount(); // Способ 2 (Только MS Windows)

QueryPerformanceCounter(&pct0); // Способ 3 (Только MS Windows)

\_\_int64 StartTscValue, EndTscValue; //Способ 4 (регистр счётчика TSC)

// Начальное значение счётчика

\_\_asm

{

RDTSC

MOV DWORD PTR [StartTscValue], EAX

MOV DWORD PTR [StartTscValue+4], EDX

}

double x;

f = (double\*)malloc(numberpoints \* sizeof(double));

x = (b-a);

//Инициализация значений нашей функции 1/x

for (i = 0; i < numberpoints; i++) {

f[i] = 1/x;

cout << " X" << i << " = " << x << fixed << setprecision(5) << "\t" << "Y" << i << " = " << f[i] << "\n";

cout << "-------------------------------------" << endl;

x = x + h;

}

cout << "\n";

Simpson(a, b, h, eps);

// Измеряем время окончания 4-мя способами

c1 = clock();

tt1 = GetTickCount();

QueryPerformanceCounter(&pct1);

// Конечное значение счетчика

\_\_asm

{

RDTSC

MOV DWORD PTR [EndTscValue], EAX

MOV DWORD PTR [EndTscValue+4], EDX

}

// Разрешаем выполнение потока на всех доступных процессорах

SetThreadAffinityMask(hThread, prevMask );

// Выводим результат измерения

printf ("\t Время измерено в секундах\n");

cout << " --------------------------------------------------------" << endl;

printf ("\t Затраченное CPU time: %f с\n", (float) (c1 - c0)/CLOCKS\_PER\_SEC);

cout << " --------------------------------------------------------" << endl;

printf ("\t Затраченное GetTickCount time: %f c\n", (tt1 - tt0));

cout << " --------------------------------------------------------" << endl;

printf ("\t Затраченное PerformanceCounter time: %f c\n", (unsigned int) ( ((double)pct1.QuadPart /(double) freq.QuadPart - (double)pct0.QuadPart / ( double) freq.QuadPart) \* 1000));

cout << " --------------------------------------------------------" << endl;

// Время выполнения

std::cout << " Затраченное time TSC: " << ((EndTscValue - StartTscValue) / frequency)<< " c\n";

std::getchar();

return 0;

}