МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**“УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Факультет информационных систем и технологий

Кафедра “ Вычислительная техника”

Дисциплина “Высокопроизводительные вычисления”

**Лабораторная работа №1**

«Исследование способов измерения затрат времени исполнения кода»

Вариант 3

Выполнил:

студент гр. ИВТАПбд-31

Вершинин Д. В.

Проверил:

Негода В.Н.

Ульяновск, 2019

**Вариант задания для my\_func**

|  |  |
| --- | --- |
| № | Функция |
| 3 | *cos x* |

**Цель работы:**

Изучение способов измерения затрат времени исполнения кода. Приобретение умений и навыков повышения точности измерения в различных условиях.

**Анализ базовых средств измерения затрат времени**

В лабораторной работе используются следующие средства измерения времени: функция clock(), высокоточный таймер HPET, счетчик тактов процессора TSC.

Функция **clock()** описывается в библиотеке time.h и возвращает число тиков от момента загрузки программы. Тик обычно равен 1 миллисекунде, но для возможности в последующем работать с другой длительностью такта в time.h фиксируется константа CLOCKS\_PER\_SEC.

**HPET** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) High Precision Event Timer, таймер событий высокой точности) — тип [таймера](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B9%D0%BC%D0%B5%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), используемый в [персональных компьютерах](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). Совместно разработан [Intel](https://ru.wikipedia.org/wiki/Intel) и [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), и стал внедряться в [чипсеты](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D0%BF%D1%81%D0%B5%D1%82) [персональных компьютеров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) с 2005 года. Старые операционные системы не поддерживают HPET и могут использовать только старые таймеры (наподобие [RTC](https://ru.wikipedia.org/wiki/RTC)). В С/С++ он используется через следующие функции, описанные в windows.h:

QueryPerformanceCounter(&tick) // прочитать номер тика таймера

QueryPerformanceFrequency(&freq) // получить частоту таймера

Основным компонентом HPET является 32-х или 64-х битный регистр, который заполняется с частотой не менее 10 МГц (обычно 14.318 или 25 МГц).

К недостаткам HPET можно отнести то, что он недоступен из большинства операционных систем, выпущенных до 2004 года.

Счетчик тактов процессора **TSC** (Time Stamp Counter) встроен в ядро CPU и обычно прирастает на 1 с каждым тактом задающего генератора. В С/С++ он используется через функцию \_\_int64 \_\_ rdtsc(), описанную в windows.h.

Аппаратно этот таймер реализован непосредственно в процессоре. Так как частота CPU в современных компьютерах намного больше, чем в других аппаратных счетчиках, то TSC обеспечивает очень высокую разрешающую способность измерения. Инструкция RDTSC позволяет получить данные счетчика в течение нескольких десятков циклов процессора, что позволяет использовать этот таймер для измерения экстремально малых промежутков времени при сравнительно низких затратах процессорных ресурсов. Значение разрешающей способности TSC напрямую зависит от тактовой частоты центрального процессора.

Минусом данного таймера является то, что перед его использованием нужно провести манипуляции по определению реальной текущей частоты процессора и опираясь на эти измерения можно перевести такты процессора в реальное время.

Исходный код из раздаточных материалов содержит:

-класс Clocks. Он является родительским для остальных таймеров. Содержит публичные свойства time и freq, в которых хранятся время и частота соответственно. Метод my\_func, в котором будет исполняться код, для которого проводятся замеры времени. Виртуальный метод measure, непосредственно измеряющий время работы кода, который будет переопределяться для каждого из таймеров. И метод masSeries, который проводит заданное количество измерений с заданной рабочей нагрузкой и выводит время каждого измерения на экран.

class Clocks { // Общее для всех видов часов

public:

string name; // название типа часов

\_\_int64 time, freq; // время и частота

void my\_func(int arcount) { // заполняет arcount элементов массива arr

return;

//for (int i = 0; i < arcount; i++) { arr[i] = sqrt(i / arcount); }

}

virtual void measure(int arcount) {}; // функция измерения времени

void measSeries(int scount, int arcount); // серия из scount измерений с рабочей нагрузкой arcount

};

void Clocks::measSeries(int scount, int arcount) {

cout << endl << name << ": " << arcount << " элементов" << endl

<< "Время (нс) \tЧастота (гц)\n";

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

measure(arcount);

cout.width(10);

cout.setf(ios::right);

cout << time << "\t" << freq << endl;

}

}

- классы Clock, QPC и TSC описывают соответственно таймеры Clock, HPET и TSC. В конструкторах устанавливаются имена каждого из таймеров, а также соответствующие частоты. Для таймера Clock частотой будет фиксированная константа CLOCK\_PER\_SEC.

class Clock : public Clocks { // часы на основе функции clock()

public:

Clock() { freq = CLOCKS\_PER\_SEC; name = "clock"; }

void measure(int sz) {

\_\_int64 t = clock(); // стартовая засечка

for (\_\_int64 i = 0; i < sz; i++)

my\_func(sz);

time = clock() - t; // замер продолжительности

}

};

//////////////////////HPET

class QPC : public Clocks { // часы на основе QPC

public:

QPC() { QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq); name = "QPC"; }//сколько раз в секунду меняется значение таймера (частота таймера)

void measure(int sz) {

LONGLONG t\_start, t\_end;

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_start); // стартовая засечка значение высокоточного системного таймера

for (\_\_int64 i = 0; i < sz; i++)

my\_func(sz);

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_end);

\_\_int64 temp = t\_end - t\_start;

time = 1000000 \* (t\_end - t\_start) / freq;

}

};

//////////////////////////RDTSC

class TSC : public Clocks { // часы на основе Time Stamp Counter

public:

TSC() { name = "TSC"; }

void measure(int sz) {

freq = hz\_cpu(); //частота процессора

\_\_int64 t = \_\_rdtsc(); // стартовая засечка

for (\_\_int64 i = 0; i < sz; i++)

my\_func(sz);

t = \_\_rdtsc() - t; // замер продолжительности

time = 1000000 \* t / freq;//мкс

}

\_\_int64 hz\_cpu() { // замер частоты

clock\_t t\_clock;

\_\_int64 t\_tsc;

t\_clock = clock() + CLOCKS\_PER\_SEC;

t\_tsc = \_\_rdtsc(); // взять TSC

while (clock() < t\_clock); // отсчет одной секунды

return (\_\_rdtsc() - t\_tsc); // частота в герцах

}

};

Проведем измерение для таймера Clock(). Проведем 20 замеров времени, чтобы посмотреть повторяемость измерений. Измерения будем проводить в двух режимах: Debug и Release для функций MyFunc() и Dummy(). MyFunc() будет заполнять массив размером DARR\_SIZE элементов, Dummy будет DARR\_SIZE раз возвращать 0. Во всех измерениях DARR\_SIZE = 100000000.

Clock cl;

cl.measSeries(20, SIZE\_DARR);

|  |  |  |  |
| --- | --- | --- | --- |
| MyFunc(), мс | | Dummy(), нс | |
| Debug | Release | Debug | Release |
| 1402 | 1218 | 28.1 | 3.33 |
| 1047 | 797 | 25.13 | 3.08 |
| 1016 | 774 | 25.05 | 2.76 |
| 1017 | 790 | 25.05 | 2.63 |
| 1022 | 800 | 24.94 | 2.59 |
| 997 | 779 | 24.92 | 2.54 |
| 1008 | 775 | 24.96 | 2.53 |
| 1005 | 779 | 24.78 | 2.53 |
| 982 | 770 | 24.75 | 2.52 |
| 970 | 759 | 24.8 | 2.59 |
| 980 | 764 | 24.93 | 2.52 |
| 985 | 758 | 24.79 | 2.56 |
| 997 | 762 | 24.97 | 2.55 |
| 1000 | 761 | 24.88 | 2.57 |
| 964 | 778 | 24.8 | 2.5 |
| 962 | 760 | 24.76 | 2.53 |
| 984 | 762 | 24.75 | 2.53 |
| 975 | 755 | 25.33 | 2.56 |
| 976 | 759 | 24.89 | 2.53 |
| 969 | 761 | 24.9 | 2.55 |

То же самое проделаем для таймера HPET.

QPC qpc;

qpc.measSeries(20, SIZE\_DARR);

|  |  |  |  |
| --- | --- | --- | --- |
| MyFunc(), мс | | Dummy(), нс | |
| Debug | Release | Debug | Release |
| 1357.51 | 1248.68 | 28.8803 | 3.03027 |
| 964.475 | 860.678 | 24.867 | 3.23886 |
| 972.623 | 795.571 | 25.0217 | 3.04875 |
| 969.545 | 791.645 | 25.1543 | 2.82481 |
| 1019.33 | 794.94 | 24.9734 | 2.81048 |
| 974.513 | 786.845 | 25.2402 | 2.92939 |
| 972.419 | 765.903 | 25.0468 | 2.63432 |
| 968.467 | 782.838 | 24.8872 | 2.52609 |
| 971.052 | 766.937 | 24.8915 | 2.53605 |
| 966.608 | 778.741 | 24.875 | 2.49898 |
| 976.228 | 769.858 | 25.1324 | 2.58033 |
| 964.038 | 772.304 | 25.0221 | 2.51382 |
| 960.939 | 764.176 | 24.9659 | 2.49876 |
| 959.763 | 764.309 | 24.8888 | 2.49745 |
| 961.056 | 765.16 | 24.789 | 2.5538 |
| 952.649 | 774.13 | 24.8559 | 2.51962 |
| 970.226 | 753.142 | 24.8943 | 2.49801 |
| 960.221 | 773.916 | 24.8805 | 2.51312 |
| 959.043 | 769.442 | 24.8635 | 2.53226 |
| 972.837 | 786.747 | 24.9267 | 2.48568 |

И для таймера TSC.

TSC tsc;

tsc.measSeries(20, SIZE\_DARR);

|  |  |  |  |
| --- | --- | --- | --- |
| MyFunc(), мс | | Dummy(), нс | |
| Debug | Release | Debug | Release |
| 1193.76 | 1134.45 | 25.3502 | 2.77839 |
| 1016.06 | 808.206 | 25.2068 | 2.51633 |
| 1018.53 | 783.665 | 25.4642 | 2.50973 |
| 1020.07 | 784.351 | 25.5042 | 2.49274 |
| 1006.49 | 787.044 | 25.1051 | 2.5306 |
| 1003.55 | 783.549 | 25.1536 | 2.55849 |
| 1020.95 | 771.858 | 25.3878 | 2.51568 |
| 1002.12 | 764.295 | 25.2619 | 2.52697 |
| 966.822 | 765.253 | 25.0572 | 2.49683 |
| 985.241 | 762.725 | 25.5139 | 2.519 |
| 967.124 | 772.27 | 25.3192 | 2.54006 |
| 970.272 | 758.853 | 25.0846 | 2.51784 |
| 971.557 | 764.44 | 25.1084 | 2.48502 |
| 965.035 | 783.563 | 25.2195 | 2.53874 |
| 963.073 | 753.682 | 25.1052 | 2.50441 |
| 971.706 | 758.952 | 25.2486 | 2.55484 |
| 964.699 | 771.724 | 25.2078 | 2.53705 |
| 953.187 | 744.918 | 25.3035 | 2.5171 |
| 968.455 | 758.504 | 25.2024 | 2.50528 |
| 958.834 | 788.712 | 25.1363 | 2.51177 |

По приведенным таблицам можно сделать следующие выводы:

- все таймеры показывают близкие значения;

- время выполнения функций в режиме “Debug” больше, чем при “Release”. Это объясняется тем, что в режиме “Debug” файл содержит дополнительную отладочную информацию, отсюда более длительное исполнение.

**2. Исследование разрешающей способности и точности измерений при использовании функции clock().**

Разрешающая способность таймера определяется посредством замера постоянно увеличивающегося числа запусков пустой функции dummy() до тех пор, пока таймер не покажет значение больше 0.

Код для функции clock():

double time;

int test\_size = 1000;

int freq = CLOCKS\_PER\_SEC;

for (int j = 0; j < 100; j++) {

\_\_int64 t = clock();

for (int i = 0; i < test\_size; i++) {

dummy();

}

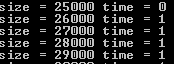
time = clock() - t;

time = (double)time \* 1000.0 / (double)freq;

cout << "size = " << test\_size << " time = " << time << endl;

test\_size += 1000;

}



Результаты измерений разрешающей способности для функции clock(), измерения времени приведены в миллисекундах. После значения size > 26000 функция возвращает значения > 1.

**Оценка линейности:**

Линейность будет оцениваться посредством запуска функции my\_func() с линейно увеличивающейся нагрузкой count, затем по точкам будет аппроксимироваться линейная функция, в результате чего будет получено линейное уравнение с коэффициентами a и b, с помощью которого можно найти “идеальные” значения для данного count.

a&=\frac{\sum x_i \sum y_i- n\sum x_iy_i}{\left(\sum x_i\right)^2-n\sum x_i^2} b&=\frac{\sum x_i \sum x_iy_i-\sum x_i^2\sum y_i}{\left(\sum x_i\right)^2-n\sum x_i^2}

Находится разница между ожидаемыми и полученными значениями, а затем и среднеквадратичное отклонение отклонений полученных величин от ожидаемых.

Код для оценки линейности:

vector<Measure> accurary\_clock\_test() {

int size = 1000000;

clock\_t t\_start, time, freq = CLOCKS\_PER\_SEC;

vector<Measure> measure;

Clocks cl;

do {

t\_start = clock();

my\_func(size);

time = clock() - t\_start;

Measure meas({ (double)time, (double)size });

measure.push\_back(meas);

size += 1000000;

} while (time <= 2020);

auto it = measure.begin();

measure.erase(it);

return measure;

}

void calculate\_accurary(vector<Measure> measure) {

double total\_difference = 0, average\_difference = 0, total\_difference\_sq = 0;

vector<double> times, counts;

for (int i = 0; i < measure.size(); i++) {

times.push\_back(measure[i].time);

counts.push\_back(measure[i].count);

}

Coefficients coeff = get\_coefficients(counts, times);

for (int i = 0; i < measure.size(); i++) {

measure[i].perfect\_time = coeff.a \* counts[i] + coeff.b;

measure[i].difference = abs(measure[i].perfect\_time - measure[i].time);

total\_difference += measure[i].difference;

}

average\_difference = total\_difference / measure.size();

for (int i = 0; i < measure.size(); i++) {

measure[i].difference\_sq = (measure[i].difference - average\_difference) \* (measure[i].difference - average\_difference);

total\_difference\_sq += measure[i].difference\_sq;

}

double s = sqrt(1.0 / (measure.size() - 1.0) \* total\_difference\_sq);

cout << "\n\ns = " << s << "\n\nXcp = " << average\_difference << "\n\n";

for (int i = 0; i < measure.size(); i++)

cout << measure[i].time << endl;

}

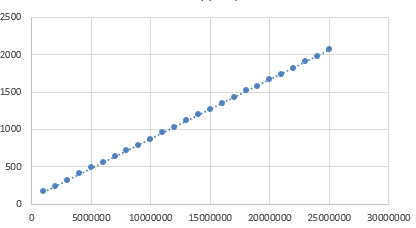
В данной таблице все временные данные представлены в миллисекундах.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | count | Изм. Время | Предп. Время | Xi | (Xi - Xср)2 |
| 0 | 2000000 | 171 | 165.2892 | 5.710769 | 0.106376 |
| 1 | 3000000 | 241 | 244.2285 | 3.228462 | 4.648999 |
| 2 | 4000000 | 324 | 323.1677 | 0.832308 | 20.72351 |
| 3 | 5000000 | 405 | 402.1069 | 2.893077 | 6.207764 |
| 4 | 6000000 | 490 | 481.0462 | 8.953846 | 12.73941 |
| 5 | 7000000 | 563 | 559.9854 | 3.014615 | 5.6169 |
| 6 | 8000000 | 635 | 638.9246 | 3.924615 | 2.1316 |
| 7 | 9000000 | 716 | 717.8638 | 1.863846 | 12.39582 |
| 8 | 10000000 | 789 | 796.8031 | 7.803077 | 5.848956 |
| 9 | 11000000 | 863 | 875.7423 | 12.74231 | 54.13564 |
| 10 | 12000000 | 956 | 954.6815 | 1.318462 | 16.53361 |
| 11 | 13000000 | 1030 | 1033.621 | 3.620769 | 3.111153 |
| 12 | 14000000 | 1116 | 1112.56 | 3.44 | 3.781529 |
| 13 | 15000000 | 1199 | 1191.499 | 7.500769 | 4.478107 |
| 14 | 16000000 | 1266 | 1270.438 | 4.438462 | 0.895207 |
| 15 | 17000000 | 1345 | 1349.378 | 4.377692 | 1.013894 |
| 16 | 18000000 | 1431 | 1428.317 | 2.683077 | 7.29831 |
| 17 | 19000000 | 1519 | 1507.256 | 11.74385 | 40.43982 |
| 18 | 20000000 | 1579 | 1586.195 | 7.195385 | 3.278885 |
| 19 | 21000000 | 1665 | 1665.135 | 0.134615 | 27.5625 |
| 20 | 22000000 | 1733 | 1744.074 | 11.07385 | 32.36735 |
| 21 | 23000000 | 1821 | 1823.013 | 2.013077 | 11.36727 |
| 22 | 24000000 | 1914 | 1901.952 | 12.04769 | 44.39659 |
| 23 | 25000000 | 1976 | 1980.892 | 4.891538 | 0.243125 |
| 24 | 26000000 | 2067 | 2059.831 | 7.169231 | 3.184852 |

Коэффициенты линейного уравнения: a = 7.89392e-05; b = 7.41077.

Xmax = 12,7423 мс; Xmin = 0,13462 мс.

Xcp = 5,38462; S = 3,67711; 1.70751< Xcp < 9.06173



Нагрузки подобраны таким образом, чтобы времена измерения входили в диапазон 20-2020 мс.

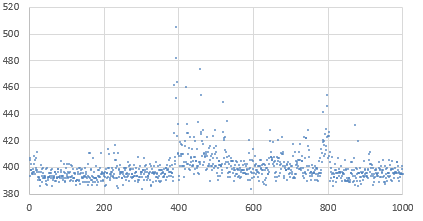
Из полученных данных и графика видно, что, действительно, представлена линейная зависимость, точки на графике лежат очень близко к своим ожидаемым значениям.

Результаты анализа повторяемости (время представлено в мс):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | | **время** | **Xi - Xср** | |
| 1 | | 406 | 37.0759 | |
| 2 | | 394 | 34.9399 | |
| 3 | | 407 | 50.2539 | |
| 4 | | 398 | 3.65192 | |
| 5 | | 398 | 3.65192 | |
| ……………………………….. | | | | |
| 997 | 401 | | | 1.18592 |
| 998 | 399 | | | 0.829921 |
| 999 | 395 | | | 24.1179 |
| 1000 | 395 | | | 24.1179 |

count = 5000000; max = 510 мс; min = 382 мс; average = 394.743 мс;

S = 11.6401 мс; 383.103 < average < 406.383



По полученным данным и построенному графику можно сказать, что практически все значения очень близки и лежат на уровне 390-400 мс, следовательно, точность достаточно высока.

**3. Исследование разрешающей способности и точности измерений при использовании таймера HPET.**

Разрешающая способность таймера определяется посредством замера постоянно увеличивающегося числа запусков пустой функции dummy() до тех пор, пока таймер не покажет значение больше 0.

double time;

int test\_size = 1;

int freq = CLOCKS\_PER\_SEC;

LONGLONG t\_start, t\_end, frequ;

for (int j = 0; j < 100; j++) {

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_start);

QueryPerformanceFrequency((LARGE\_INTEGER \*)&frequ);

for (int i = 0; i < test\_size; i++)

dummy();

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_end);

time = t\_end - t\_start;

time = 1000000000.0 \* time / frequ;

cout << "size = " << test\_size << " time = " << time << endl;

test\_size++;

}



Значения отличные от 0 таймер HPET начинает выдавать постоянно при count больше, чем 5. Однако, видно, что выдаются одни и те же значения это связано с тем, что один тик таймера происходит за 410.546 нс.

Код для оценки линейности:

vector<Measure> accurary\_qpc\_test() {

int size = 125;

\_\_int64 t\_start, t\_end, time, freq; vector<Measure> measure;

QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq);

do {

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_start);

my\_func(size);

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_end);

time = t\_end - t\_start;

time = 1000000.0 \* time / freq;

cout << "size = " << size << " :time " << time << "\n";

Measure meas({ (double)time, (double)size });

measure.push\_back(meas);

size += 125;

} while (time <= 202);

return measure;

}

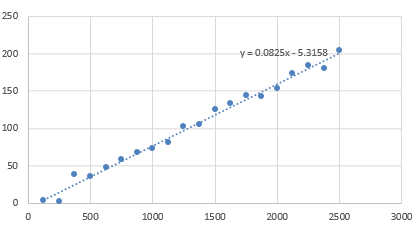
Оценка линейности (время представлено в мкс):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | count | Изм. Время | Предп. Время | Xi | (Xi - Xср) |
| 0 | 125 | 4 | 5 | 1 | 16.85319 |
| 1 | 250 | 2 | 15.315789 | 13.31579 | 67.41274 |
| 2 | 375 | 38 | 25.631579 | 12.36842 | 52.75346 |
| 3 | 500 | 36 | 35.947368 | 0.052632 | 25.52909 |
| 4 | 625 | 48 | 46.263158 | 1.736842 | 11.34626 |
| 5 | 750 | 59 | 56.578947 | 2.421053 | 7.204986 |
| 6 | 875 | 68 | 66.894737 | 1.105263 | 16 |
| 7 | 1000 | 73 | 77.210526 | 4.210526 | 0.800554 |
| 8 | 1125 | 81 | 87.526316 | 6.526316 | 2.019391 |
| 9 | 1250 | 103 | 97.842105 | 5.157895 | 0.00277 |
| 10 | 1375 | 105 | 108.157895 | 3.157895 | 3.792244 |
| 11 | 1500 | 126 | 118.473684 | 7.526316 | 5.861496 |
| 12 | 1625 | 133 | 128.789474 | 4.210526 | 0.800554 |
| 13 | 1750 | 144 | 139.105263 | 4.894737 | 0.044321 |
| 14 | 1875 | 143 | 149.421053 | 6.421053 | 1.731302 |
| 15 | 2000 | 154 | 159.736842 | 5.736842 | 0.398892 |
| 16 | 2125 | 174 | 170.052632 | 3.947368 | 1.34072 |
| 17 | 2250 | 184 | 180.368421 | 3.631579 | 2.171745 |
| 18 | 2375 | 180 | 190.684211 | 10.68421 | 31.12465 |
| 19 | 2500 | 205 | 201 | 4 | 1.221607 |

Коэффициенты линейного уравнения: a = 0,0825263, b = -5,31579.

Xmax = 13,3158 мкс; Xmin = 0,05263 мкс.

Xcp = 5,10526; S = 3,61583; 1,48943 < Xcp < 8,72109



Нагрузки count подобраны таким образом, чтобы времена измерения входили в диапазон 2-202 мкс.

Из полученных данных и графика видно, что, действительно, представлена линейная зависимость. Некоторые точки на графике не лежат на прямой, а несколько отконены. Можно предположить, что это происходит из-за обращений к кэш-памяти, а именно когда нужных данных в кэш-памяти не обнаруживается, процессор простаивает, ожидая их получения из ОЗУ.

**Анализ повторяемости:**

Для этого необходимо выбрать нагрузку count таким образом, чтобы замеры попадали в диапазон 2-202 мкс. Проводится 1000 измерений, находится максимальное, минимальное и среднее значения, а также определяется среднеквадратичное отклонение.

Код для анализа повторяемости (время представлено в мкс):

void check\_repeatability\_QPC() {

int size = 1500;

int count\_of\_repeats = 1000;

double max\_time = 0, min\_time = 10000, average\_time, total\_average\_time = 0, total\_difference\_sq = 0;

vector<double> times, difference\_sq;

\_\_int64 t\_start, t\_end, freq, time;

QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq);

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

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_start);

my\_func(size);

QueryPerformanceCounter((LARGE\_INTEGER \*)&t\_end);

time = t\_end - t\_start;

time = 1000000.0 \* time / freq;

times.push\_back(time);

if (max\_time < time) max\_time = time;

if (min\_time > time) min\_time = time;

total\_average\_time += time;

}

average\_time = total\_average\_time / count\_of\_repeats;

for (int i = 0; i < times.size(); i++) {

difference\_sq.push\_back((average\_time - times[i]) \* (average\_time - times[i]));

total\_difference\_sq += difference\_sq[i];

}

double s = sqrt(1.0 / (count\_of\_repeats - 1.0) \* total\_difference\_sq);

cout << "count=" << size << "\nmax=" << max\_time << "\nmin=" << min\_time << "\naverage=" <<

average\_time << "\ns=" << s << "\ns+-={" << average\_time - s << ":" << average\_time + s << "}\n\n";

cout << "i time difference\_sq\n";

for (int i = 0; i < times.size(); i++)

cout << i + 1 << " " << times[i] << " " << difference\_sq[i] << "\n";

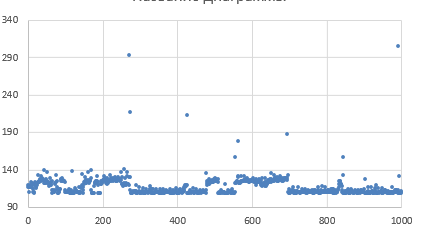
}

|  |  |  |
| --- | --- | --- |
| i | Изм. Время | (Xi - Xср) |
| 1 | 119 | 6.37563 |
| 2 | 117 | 0.275625 |
| 3 | 117 | 0.275625 |
| 4 | 110 | 41.9256 |
| 5 | 117 | 0.275625 |
| ……………… | | |
| 997 | 108 | 71.8256 |
| 998 | 109 | 55.8756 |
| 999 | 111 | 29.9756 |
| 1000 | 110 | 41.9256 |

count = 1500;

Xmax = 305 мкс; Xmin = 107 мкс; Xaverage = 116,475 мкс;

S = 12,7243 мкс; 103,751 < average < 129,199



По полученным данным и построенному графику можно сказать, что большая часть значений близка и находится на уровне 110-120 мс.

**4. Исследование разрешающей способности и точности измерений при использовании таймера RDTSC.**

Разрешающая способность таймера определяется посредством замера постоянно увеличивающегося числа запусков пустой функции dummy() до тех пор, пока таймер не покажет значение больше 0.

Код для таймера RDTSC:

double time;

int test\_size = 1;

LONGLONG t\_start, t\_end, frequ;

for (int j = 0; j < 100; j++) {

t\_start = \_\_rdtsc();

for (int i = 0; i < test\_size; i++)

dummy();

time = \_\_rdtsc() - t\_start;

frequ = tsc.hz\_cpu();

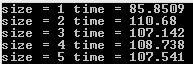
time = 1000000000.0 \* time / frequ;

cout << "size = " << test\_size << " time = " << time << endl;

test\_size++;

}

Результаты измерения разрешающей способности таймера RDTSC, временные затраты представлены в нс.



Значения отличные от 0 таймер RDTSC начинает выдавать постоянно сразу же, т.е. при count = 1 и больше.

Код для оценки линейности:

vector<Measure> accurary\_rdtsc\_test() {

int size = 1;

\_\_int64 t\_start;

double freq, time;

vector<Measure> measure;

do {

t\_start = \_\_rdtsc();

my\_func(size);

time = \_\_rdtsc() - t\_start;

freq = hz\_cpu();

time = 1000000000.0 \* time / freq;

cout << "size = " << size << ": time = " << time << "\n";

Measure meas({ (double)time, (double)size });

measure.push\_back(meas);

size += 1;

} while (time <= 2200);

return measure;

}

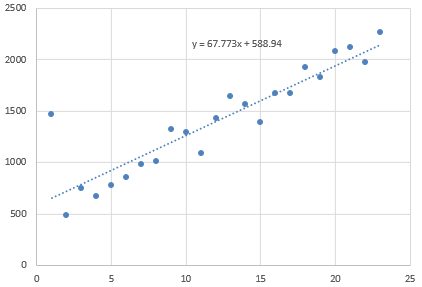
Оценка линейности (все данные представлены в нс):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | count | Изм. Время | Предп. Время | Xi | (Xi - Xср) |
| 0 | 1 | 1464.7 | 656.71 | 807.945 | 440364.1 |
| 1 | 2 | 481.24 | 724.48 | 243.2488 | 9781.82 |
| 2 | 3 | 750.67 | 792.26 | 41.58335 | 10560.1 |
| 3 | 4 | 666.8 | 860.03 | 193.2354 | 2390.202 |
| 4 | 5 | 779.76 | 927.8 | 148.0474 | 13.70233 |
| 5 | 6 | 851.83 | 995.58 | 143.7445 | 0.361465 |
| 6 | 7 | 979.5 | 1063.4 | 83.85345 | 3659.313 |
| 7 | 8 | 1007.2 | 1131.1 | 123.9138 | 417.4619 |
| 8 | 9 | 1323.1 | 1198.9 | 124.1584 | 407.5283 |
| 9 | 10 | 1290.1 | 1266.7 | 23.47459 | 14609.83 |
| 10 | 11 | 1092.6 | 1334.4 | 241.8133 | 9499.94 |
| 11 | 12 | 1425.8 | 1402.2 | 23.63081 | 14572.08 |
| 12 | 13 | 1645.2 | 1470 | 175.1959 | 951.7377 |
| 13 | 14 | 1564.8 | 1537.8 | 27.04467 | 13759.53 |
| 14 | 15 | 1385.7 | 1605.5 | 219.8556 | 5701.738 |
| 15 | 16 | 1672 | 1673.3 | 1.316125 | 20457.46 |
| 16 | 17 | 1673.5 | 1741.1 | 67.56626 | 5895.082 |
| 17 | 18 | 1926 | 1808.9 | 117.1642 | 738.8314 |
| 18 | 19 | 1830.1 | 1876.6 | 46.49901 | 9573.974 |
| 19 | 20 | 2079.9 | 1944.4 | 135.4617 | 78.9261 |
| 20 | 21 | 2122.7 | 2012.2 | 110.5304 | 1143.477 |
| 21 | 22 | 1974.7 | 2080 | 105.2986 | 1524.673 |
| 22 | 23 | 2263.1 | 2147.7 | 115.3699 | 839.5985 |

Коэффициенты линейного уравнения: a =67.7734; b =588.938.

Xmax = 243.2488 нс; Xmin = 1.316125 нс.

Xcp = 144,346; S =160,541; 16,195< Xcp < 304,887



Нагрузки count подобраны таким образом, чтобы времена измерения входили в диапазон 200-2220 нс.

Из полученных данных и графика видно, что, действительно, представлена линейная зависимость, однако точки на графике не лежат на одной прямой, а, следовательно, не совпадают со своими ожидаемыми значениями. Вероятно, что это происходит из-за предсказателя переходов (прогнозирование ветвлений) – устройства, входящего в состав процессора, предсказывающего, будет ли выполнен условный переход в исполняемой программе.

**Анализ повторяемости:**

Для этого необходимо выбрать нагрузку count таким образом, чтобы замеры попадали в диапазон 200-2200 нс. Проводится 1000 измерений, находится максимальное, минимальное и среднее значения, а также определяется среднеквадратичное отклонение.

Код для анализа повторяемости (время представлено в нс):

void check\_repeatability\_TSC() {

int size = 15;

int count\_of\_repeats = 1000;

double max\_time = 0, min\_time = 10000, average\_time, total\_average\_time = 0, total\_difference\_sq = 0;

vector<double> times, difference\_sq;

\_\_int64 t\_start, t\_end, freq, time;

freq = hz\_cpu();

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

t\_start = \_\_rdtsc();

my\_func(size);

t\_end = \_\_rdtsc();

time = t\_end - t\_start;

time = 1000000000.0 \* time / freq;

times.push\_back(time);

if (max\_time < time) max\_time = time;

if (min\_time > time) min\_time = time;

total\_average\_time += time;

}

average\_time = total\_average\_time / count\_of\_repeats;

for (int i = 0; i < times.size(); i++) {

difference\_sq.push\_back((average\_time - times[i]) \* (average\_time - times[i]));

total\_difference\_sq += difference\_sq[i];

}

double s = sqrt(1.0 / (count\_of\_repeats - 1.0) \* total\_difference\_sq);

cout << "count=" << size << "\nmax=" << max\_time << "\nmin=" << min\_time << "\naverage=" <<

average\_time << "\ns=" << s << "\ns+-={" << average\_time - s << ":" << average\_time + s << "}\n\n";

cout << "i time difference\_sq\n";

for (int i = 0; i < times.size(); i++)

cout << i + 1 << " " << times[i] << " " << difference\_sq[i] << "\n";

}

Результаты анализа повторяемости (время в нс):

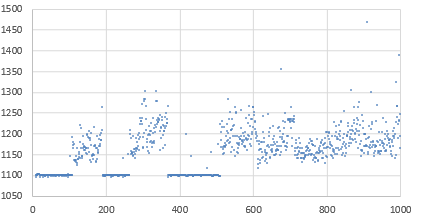
|  |  |  |
| --- | --- | --- |
| i | Изм. время | (Xi – Xср)2 |
| 1 | 13520 | 1.52E+08 |
| 2 | 2438 | 1.59E+06 |
| 3 | 2523 | 1.81E+06 |
| 4 | 1755 | 331860 |
| 5 | 1822 | 413543 |
| ………………. | | |
| 997 | 1390 | 44551.8 |
| 998 | 1246 | 4498.79 |
| 999 | 1193 | 198.049 |
| 1000 | 1165 | 193.961 |

count = 15;

max =13520; min = 1094; average = 1178,93;

S =404,614;

774,313 < average < 1583,54



Можно заметить, что разброс значений получился достаточно большим. Для решения этой проблемы интегрируем фильтр минимума в код, который будет выбирать минимальное значение для замера:

time\_min\_func = 10000;

for (int j = 0; j < 15; j++) {

t\_start = \_\_rdtsc();

my\_func(size);

time = \_\_rdtsc() - t\_start;

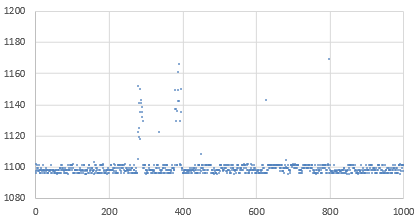
time = 1000000000.0 \* time / freq;

if (time\_min\_func > time)

time\_min\_func = time;

time = time\_min\_func;

}



В результате разброс значительно уменьшился, видно, что большая часть значений находится на уровне 1100 нс.

**Вывод**

В результате данной лабораторной работы был проведен анализ базовых средств измерения временных затрат (clock(), HPET, RDTSC), а также измерены разрешающая способность, линейность и точность измерений для каждого из таймеров.