ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2  
«Оценка производительности процессора»

по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-811  
Адов Артем Сергеевич

Проверил: ст. преп. Кафедры ВС   
Ткачёва Татьяна Алексеевна

Новосибирск 2020

Содержание:

Постановка задачи……………………………………………………………………**3**

Выполнение работы…….……………………………………………….……………**5**

Результат работы……………………………………………………………...………**6**

Приложение……………………………………………...................................……..**10**

Вывод………………………………………………………………………………...**18**

# 

# Постановка задачи

**Задание:** Реализовать программу для оценки производительности процессора (benchmark).

1. Написать программу(ы) (benchmark) на языке С/С++/C# для оценки производительности процессора. В качестве набора типовых задач использовать либо минимум 3 функции выполняющих математические вычисления, либо одну функцию по работе с матрицами и векторами данных с несколькими типами данных. Можно использовать готовые функции из математической библиотеки (math.h) [3], библиотеки BLAS [4] (англ. Basic Linear Algebra Курс «Архитектура вычислительных систем».

Subprograms — базовые подпрограммы линейной алгебры) и/или библиотеки LAPACK [5] (LinearAlgebra PACKage). Обеспечить возможность подать на вход программы общее число испытаний для каждой типовой задачи (минимум 10). Входные данные для типовой задачи сгенерировать случайным образом.

2. С помощью системного таймера (библиотека time.h, функции clock() илиgettimeofday()) или с помощью процессорного регистра счетчика TSC реализовать оценку в секундах среднего времени испытания каждой типовой задачи. Оценить точность и погрешность (абсолютную и относительную) измерения времени (рассчитать дисперсию и среднеквадратическое отклонение).

3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV со следующей структурой: [PModel;Task;OpType;Opt;LNum;InsCount;Timer;AvTime;AbsErr;RelErr;TaskPerf], где

PModel – Processor Model, модель процессора, на котором проводятся испытания;

Task – название выбранной типовой задачи (например, sin, log, saxpy, dgemv, sgemm и др.);

OpType – Operand Type, тип операндов используемых при вычислениях типовой задачи;

Opt – Optimisations, используемы ключи оптимизации (None, O1, O2 и др.);

LNum – Launch Numer, число испытаний типовой задачи.

InsCount – Instruction Count, оценка числа инструкций при выполнении типовой задачи;

AvTime – Average Time, среднее время выполнения типовой задачи в секундах;

AbsError – Absolute Error, абсолютная погрешность измерения времени в секундах;

RelError – Relative Error, относительная погрешность измерения времени в %;

TaskPerf – Task Performance, производительность (быстродействие) процессора при выполнении типовой задачи.

# Выполнение работы

# Типовые задачи:

# function1 – функция, выполняющая вычисление sin(x).

# function2 – функция, выполняющая вычисление cos(x).

# function3 – функция, выполняющая вычисление tan(x).

# Испытания:

# Входные данные генерируются случайным образом. Количество испытаний 10.

Оценка среднего времени испытания каждой типовой задачи реализуется с помощью функции clock() библиотеки ctime.

Оценка точности проводится по формуле дисперсии:

Где Х – это массив времени испытания каждой типовой задачи.

Погрешность:

* Абсолютная – среднеквадратичное отклонение, т.е
* Относительная

Где k – среднее время выполнения одной типовой задачи.

# Результат работы

# 

# 

Рисунок 1. Результат записи в файл table.csv

# 

# https://sun9-19.userapi.com/PlO5lbDKIupvoXEP1YgielHgZY7j_5Junq0vOg/PXks781gG_0.jpg

# Рис.2 «Диаграмма с ключом None»

# https://sun1-28.userapi.com/ZMLV7B7dBjx7SgP11C45b66KOsDIfWyYUBPRyQ/gSxPuPyaVvU.jpg

# Рис.3 «Диаграмма с ключом О1»

# https://sun9-22.userapi.com/KYZo9i4XrRHa93LdkRbP0EX8oGGnmvw8pVNBsw/D14i2xpNSqs.jpg

# Рис.4 «Диаграмма с ключом О2»

# https://sun9-3.userapi.com/GYIZAvSiEGuFFpTJZ_IxaHeAXYz6cib6I8rjUw/5aaCZWMO3N4.jpg

# Рис.5 «Диаграмма с ключом О3»

# Приложение

# acs\_2.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define ERROR\_OPEN\_TABLE\_FILE 0b00000001

#define ERROR\_OPEN\_TEMP\_FILE 0b00000010

#define MAXIMUM\_LENGTH\_KEY\_OPTIMIZATION\_NAME 4

#define COEFFICIENT\_FOR\_RAND 10

#define COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT 100

#define QUANTITY\_ITERATION\_FOR\_ONE\_TEST 10000

#define FILE\_NAME\_TABLE\_CVS "table.csv"

#define FILE\_NAME\_TEMP\_KEY "temp\_key.txt"

#define ADDITIONAL\_RECORDING "a"

#define READ\_ONLY "rt"

int Test\_By\_Sinus\_Int()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = rand()%COEFFICIENT\_FOR\_RAND;

time\_before\_work = clock();

result = sin(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Sinus\_Float()

{

double result = 0.0;

float test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((float)(rand()%COEFFICIENT\_FOR\_RAND) / (float)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = sin(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Sinus\_Double()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((double)(rand()%COEFFICIENT\_FOR\_RAND) / (double)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = sin(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Cosinus\_Int()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = rand()%COEFFICIENT\_FOR\_RAND;

time\_before\_work = clock();

result = cos(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Cosinus\_Float()

{

double result = 0.0;

float test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((float)(rand()%COEFFICIENT\_FOR\_RAND) / (float)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = cos(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Cosinus\_Double()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((double)(rand()%COEFFICIENT\_FOR\_RAND) / (double)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = cos(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Tangens\_Int()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = rand()%COEFFICIENT\_FOR\_RAND;

time\_before\_work = clock();

result = tan(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Tangens\_Float()

{

double result = 0.0;

float test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((float)(rand()%COEFFICIENT\_FOR\_RAND) / (float)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = tan(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

int Test\_By\_Tangens\_Double()

{

double result = 0.0;

int test\_value = 0;

int time\_before\_work = 0;

int time\_work = 0;

for(short int i = 0; i < QUANTITY\_ITERATION\_FOR\_ONE\_TEST; i++)

{

test\_value = ((double)(rand()%COEFFICIENT\_FOR\_RAND) / (double)(rand()%COEFFICIENT\_FOR\_RAND));

time\_before\_work = clock();

result = tan(test\_value);

time\_work = time\_work + (clock() - time\_before\_work);

}

return time\_work;

}

double Calculate\_Dispersion(double\* array\_time, int size\_array)

{

double dispersion\_value = 0.0;

double mean = 0.0;

for(short int i = 0; i < size\_array; i++)

{

mean += array\_time[i];

}

mean /= size\_array;

for(short int i = 0; i < size\_array; i++)

{

dispersion\_value = (dispersion\_value + pow((array\_time[i] - mean), 2));

}

dispersion\_value = (dispersion\_value / (size\_array - 1));

return dispersion\_value;

}

void Print\_Array(double\* arr, int size\_arr)

{

for(short int i = 0; i < size\_arr; i++)

{

printf("%f ",arr[i]);

}

}

char\* Get\_Key\_Optimization(char\* key\_optimization)

{

FILE\* file = NULL;

key\_optimization = (char\*) calloc (MAXIMUM\_LENGTH\_KEY\_OPTIMIZATION\_NAME,sizeof(char));

file = fopen(FILE\_NAME\_TEMP\_KEY,READ\_ONLY);//get key optimiz

if(file == NULL)

{

return NULL;

}

for(short int i = 0; ((i < 4) && (!feof(file))); i++)

{

key\_optimization[i] = getc(file);

if(key\_optimization[i] == '\n')

{

key\_optimization[i] = '\0';

break;

}

}

fclose(file); //get key optimiz // return key optimiz

return key\_optimization;

}

int main()

{

FILE\* file = NULL;

double\* array\_time = NULL;

char\* key\_optimization = NULL;

char\* system\_command = NULL;

double performance = 0.0;

double time\_before\_test = 0.0;

double average\_time = 0.0;

double accuracy = 0.0;

double absolute\_oversight = 0.0;

double relativity\_oversight = 0.0;

int error\_flag = 0b0;

int quantity\_test = 0;

srand(time(NULL));

printf("Please write need quantity tests\n");

scanf("%d",&quantity\_test);

key\_optimization = Get\_Key\_Optimization(key\_optimization);

if(key\_optimization == NULL)

{

return ERROR\_OPEN\_TEMP\_FILE;

}

file = fopen(FILE\_NAME\_TABLE\_CVS,ADDITIONAL\_RECORDING);

if(file == NULL)

{

return ERROR\_OPEN\_TABLE\_FILE;

}

array\_time = (double\*) calloc (quantity\_test, sizeof(double));

asprintf(&system\_command,"lscpu | grep 'Ð˜Ð¼Ñ' | awk '{for(i = 3; i <= NF; i++){printf $i; printf(\" \");}}' >> table.csv");

//sin int

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Sinus\_Int();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| sin | int | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//sin float

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Sinus\_Float();

average\_time = average\_time + array\_time[i];;

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| sin | float | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//sin double

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Sinus\_Double();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| sin | double | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//cos int

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Cosinus\_Int();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| cos | int | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//cos float

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Cosinus\_Float();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| cos | float | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//cos double

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Cosinus\_Double();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| cos | double | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//tan int

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Tangens\_Int();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| tan | int | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//tan float

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Tangens\_Float();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| tan | float | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

average\_time = 0.0;

performance = 0.0;

//tan double

for(short int i = 0; i < quantity\_test; i++)

{

array\_time[i] = Test\_By\_Tangens\_Double();

average\_time = average\_time + array\_time[i];

}

average\_time = (average\_time / quantity\_test);

performance = (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / (average\_time / CLOCKS\_PER\_SEC));

accuracy = Calculate\_Dispersion(array\_time, quantity\_test);

absolute\_oversight = sqrt(accuracy);

relativity\_oversight = (absolute\_oversight / average\_time \* COEFFICIENT\_FOR\_CONVERT\_IN\_PERCENT);

fprintf(file,"| ");

fflush(file);

system(system\_command);

fprintf(file,"| tan | double | %s | %d | %d | clock() | %lf | %lf | %lf | %lf |\n",

key\_optimization, quantity\_test, (QUANTITY\_ITERATION\_FOR\_ONE\_TEST / quantity\_test),

(average\_time / CLOCKS\_PER\_SEC), (absolute\_oversight / CLOCKS\_PER\_SEC), (relativity\_oversight / CLOCKS\_PER\_SEC), performance);

fflush(file);

//

fclose(file);

free(array\_time);

array\_time = NULL;

return error\_flag;

}

# acs\_2.sh

#!/bin/bash

rm temp\_key.txt;

touch temp\_key.txt;

echo "None" >> temp\_key.txt;

gcc -O0 -Wall -o key acs\_2.c -lm;

./key;

rm temp\_key.txt;

touch temp\_key.txt;

echo "-O1" >> temp\_key.txt;

gcc -O1 -Wall -o key acs\_2.c -lm;

./key;

rm temp\_key.txt;

touch temp\_key.txt;

echo "-O2" >> temp\_key.txt;

gcc -O2 -Wall -o key acs\_2.c -lm;

./key;

rm temp\_key.txt;

touch temp\_key.txt;

echo "-O3" >> temp\_key.txt;

gcc -O3 -Wall -o key acs\_2.c -lm;

./key;

rm temp\_key.txt;

rm key;

Вывод:

Быстродействие на прямую зависит от типа данных (чем более легковесный тип данных – тем менее ресурсозатратны операции с ним) и оптимизации применяемой к программе на этапе компиляции (как правило при слабой оптимизации или отсутствии таковой, программы выполняется медленно) .