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. ИА-832

Тиванов Данил Евгеньевич

Проверил: ст. преп. Кафедры ВС   
Ткачёва Татьяна Алексеевна

Новосибирск 2020

Содержание

[Постановка задачи 3](#_Toc50517127)

[Выполнение работы 5](#_Toc50517128)

[Результат работы 6](#_Toc50517129)

[Приложение](#_Toc50517130) 9

# Постановка задачи

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 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() или gettimeofday()) 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.

3. \* Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).

3. \*\* Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).

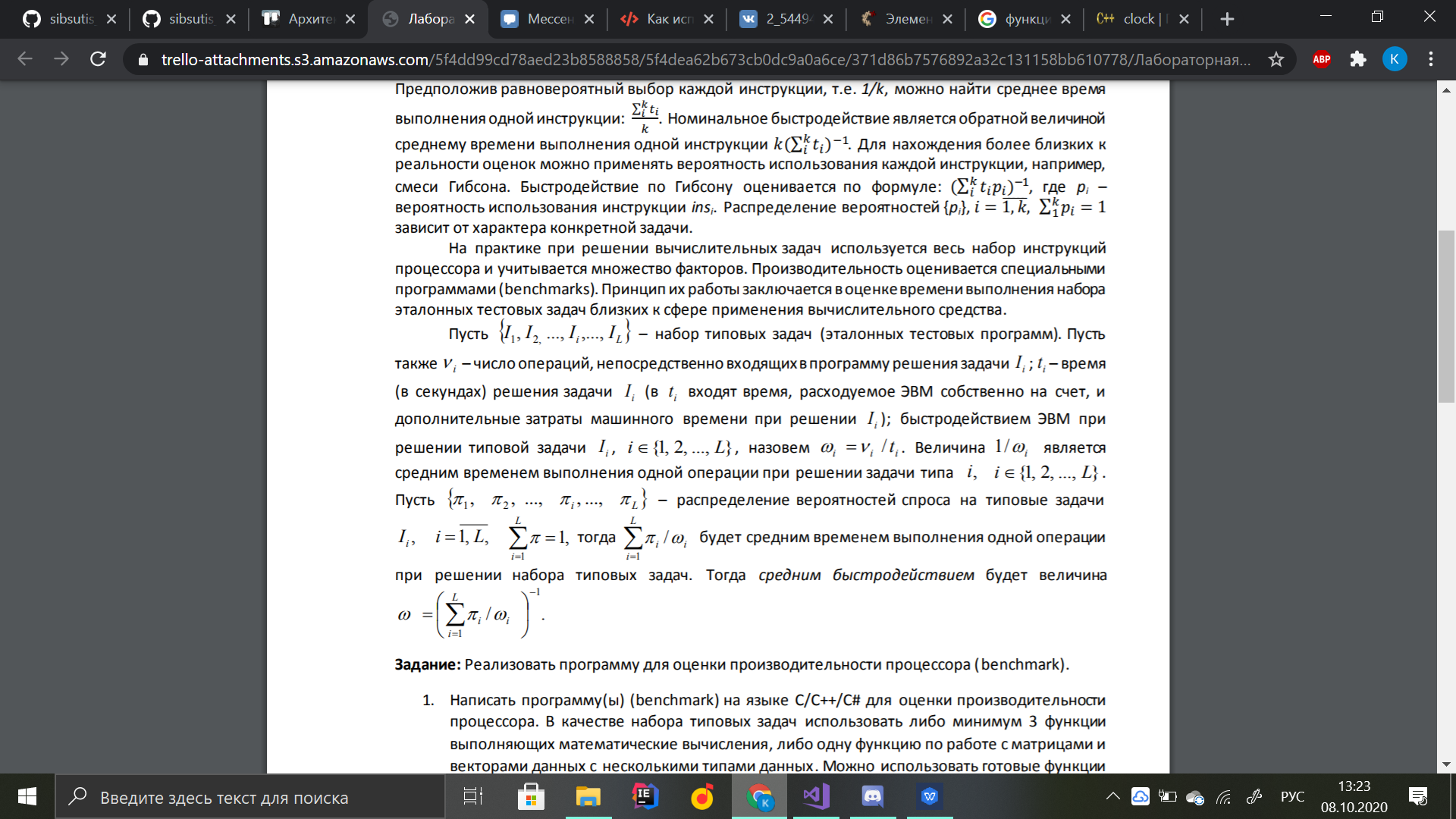
3. \*\*\* Оценить и постараться минимизировать накладные расходы(время на вызов функций, влияние загрузки системы и т.п.) при испытании, то есть добиться максимальной точности измерений.

4. Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

# Выполнение работы

В качестве типовых задач были выбраны функции библиотеки math.h: sin, cos, sqrt. При запуске пользователю дается возможность задать количество испытаний для каждой типовой задачи.

Результаты испытаний с разными типами входных данных заносятся в файл result.csv. Тип процессора определяется с помощью функции GetSystemInfo winAPI. Исходя из того, что хранится в поле dwProcessorType, можно узнать тип процессора. Для подсчета времени выполнения типовой задачи используется функция clock(), которая возвращает время, пройденное с момента запуска программы, вызвавшей функцию clock(). Для перевода времени в секунды используется константа CLOCKS\_PER\_SEC. Абсолютная и относительная погрешность вычисляются с помощью дисперсии. Среднее быстродействие вычисляется с помощью формулы:



Так же были проведены испытания типовых задач и производительности с оптимизирующими ключами -O1 (уменьшает размер кода) -O2 (создает быстрый код). Результаты были записаны в файл.

Использованы fixed (показывает, что установленная точность относится к количеству знаков после запятой) и scientific (выводит число в экспоненциальной форме) для десятичной записи в файл.

# Результат работы

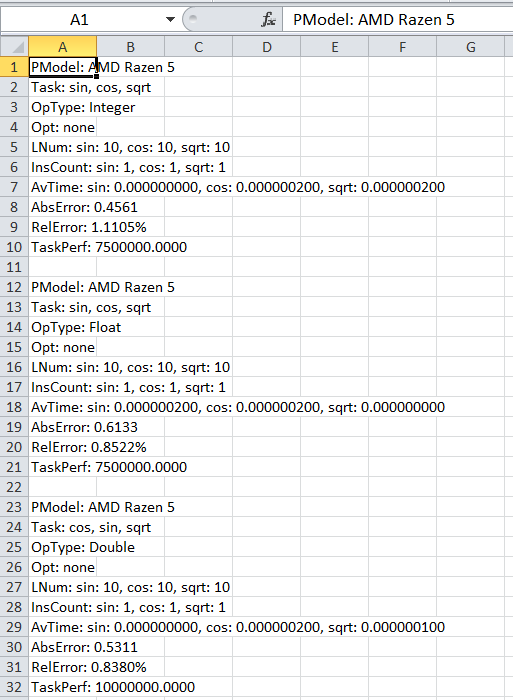
Результат работы программы без использования ключа оптимизации представлен на рисунке 1.  


Рис.1 - результат работы программы без использования ключа оптимизации при равновероятном использовании типовых задач и равном 10

Результат работы программы с оптимизирующим ключом -O1 представлен на рисунке 2.

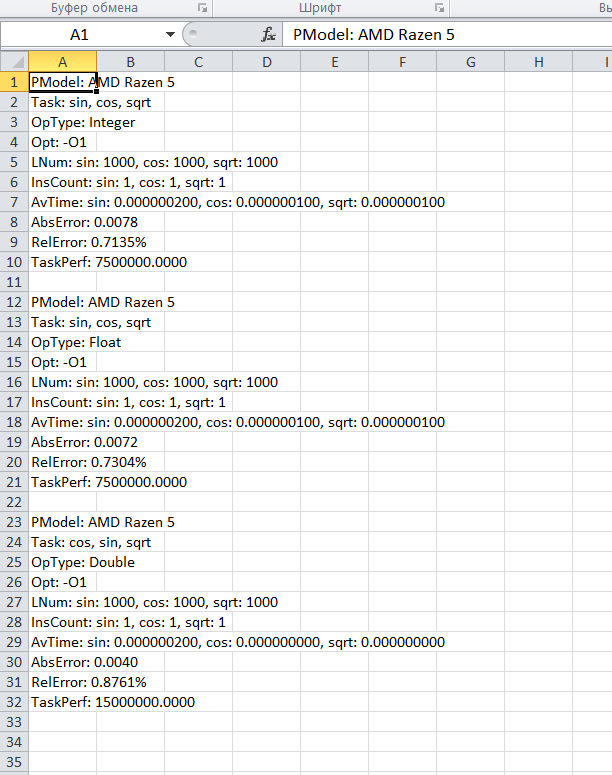


Рис.2 - результат работы программы с использованием ключа оптимизации -O1 при равновероятном использовании типовых задач и равном 1000

Результат работы программы с оптимизирующим ключом -O2 представлен на рисунке 3.

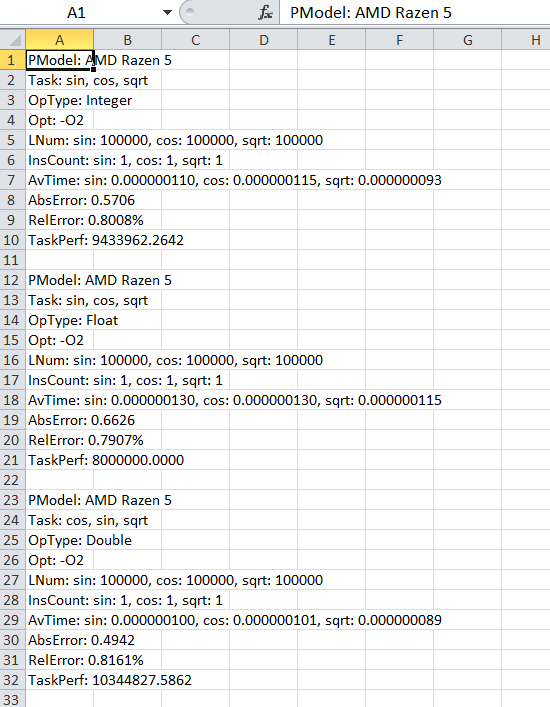


Рис.3 - результат работы программы с использованием ключа оптимизации -O2 при равновероятном использовании типовых задач и равном 100000

# Приложение

#include <iostream>

#include <math.h>

#include <time.h>

#include <Windows.h>

#include <fstream>

#include <string>

#include <iomanip>

using namespace std;

int main()

{

srand(time(0));

int countSin, countCos, countSqrt, sum;

string key;

cout << "Enter optimization key: ";

cin >> key;

SYSTEM\_INFO SystemInfo;

GetSystemInfo(&SystemInfo);

clock\_t timeT;

long double sinTime = 0, cosTime = 0, sqrtTime = 0;

double dis, m = 0, m2 = 0, t = 0;

string processor;

switch (SystemInfo.dwProcessorType)

{

case 386:

processor = "Intel 386";

break;

case 486:

processor = "Intel 486";

break;

case 586:

processor = "Intel Pentium";

break;

case 2200:

processor = "Intel IA64";

break;

case 8664:

processor = "AMD X8664";

break;

default:

processor = "Unknow processor";

}

cout << "Number of operations with sin: ";

cin >> countSin;

cout << "Number of operations with cos: ";

cin >> countCos;

cout << "Number of operations with sqrt: ";

cin >> countSqrt;

sum = countSin + countCos + countSqrt;

const int N = (countSin <= 100 && countCos <= 100 && countSqrt <= 100) ? 1000 : 10;

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

timeT = clock();

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

sin(rand());

}

sinTime += clock() - timeT;

timeT = clock();

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

cos(rand());

}

cosTime += clock() - timeT;

timeT = clock();

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

sqrt(rand());

}

sqrtTime += clock() - timeT;

t += sinTime / CLOCKS\_PER\_SEC + cosTime / CLOCKS\_PER\_SEC + sqrtTime / CLOCKS\_PER\_SEC;

m += t / N;

m2 += pow(t, 2) / N;

}

sinTime = sinTime / N / CLOCKS\_PER\_SEC;

cosTime = cosTime / N / CLOCKS\_PER\_SEC;

sqrtTime = sqrtTime / N / CLOCKS\_PER\_SEC;

dis = (double)m2 - pow(m, 2);

ofstream out("result.csv");

if (out.is\_open()) {

out << "PModel: " << processor << endl;

out << "Task: " << "sin, cos, sqrt" << endl;

out << "OpType: " << "Integer" << endl;

out << "Opt: " << key << endl;

out << "LNum: " << "sin: " << countSin << ", cos: " << countCos << ", sqrt: " << countSqrt << endl;

out << "InsCount: " << "sin: 1, cos: 1, sqrt: 1" << endl;

out << "AvTime: " << "sin: " << fixed << setprecision(9) << sinTime/ countSin << ", cos: " << cosTime / countCos << ", sqrt: " << sqrtTime / countSqrt << endl;

out << "AbsError: " << fixed << setprecision(4) << sqrt(dis)<< endl;

out << "RelError: " << sqrt(dis) / m << "%" << endl;

out << "TaskPerf: " << (double)1 / (((double)countSin / sum / (countSin / sinTime)) + ((double)countCos / sum / (countCos / cosTime)) + ((double)countSqrt / sum / (countSqrt / sqrtTime))) << endl << endl;

m = 0, m2 = 0, dis, sinTime = 0, cosTime = 0, sqrtTime = 0, t = 0;

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

timeT = clock();

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

sinf((float)rand() / RAND\_MAX);

}

sinTime += clock() - timeT;

timeT = clock();

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

cosf((float)rand() / RAND\_MAX);

}

cosTime += clock() - timeT;

timeT = clock();

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

sqrtf((float)rand() / RAND\_MAX);

}

sqrtTime += clock() - timeT;

t += sinTime / CLOCKS\_PER\_SEC + cosTime / CLOCKS\_PER\_SEC + sqrtTime / CLOCKS\_PER\_SEC;

m += t / N;

m2 += pow(t, 2) / N;

}

dis = (double)m2 - pow(m, 2);

sinTime = sinTime / N / CLOCKS\_PER\_SEC;

cosTime = cosTime / N / CLOCKS\_PER\_SEC;

sqrtTime = sqrtTime / N / CLOCKS\_PER\_SEC;

out << "PModel: " << processor << endl;

out << "Task: " << "sin, cos, sqrt" << endl;

out << "OpType: " << "Float" << endl;

out << "Opt: " << key << endl;

out << "LNum: " << "sin: " << countSin << ", cos: " << countCos << ", sqrt: " << countSqrt << endl;

out << "InsCount: " << "sin: 1, cos: 1, sqrt: 1" << endl;

out << "AvTime: " << "sin: " << fixed << setprecision(9) << sinTime / countSin<< ", cos: " << cosTime / countCos << ", sqrt: " << sqrtTime / countSqrt << endl;

out << "AbsError: " << fixed << setprecision(4) << sqrt(dis) << endl;

out << "RelError: " << sqrt(dis) / m << "%" << endl;

out << "TaskPerf: " << (double)1 / (((double)countSin / sum / (countSin / sinTime)) + ((double)countCos / sum / (countCos / cosTime)) + ((double)countSqrt / sum / (countSqrt / sqrtTime))) << endl << endl;

m = 0, m2 = 0, dis, sinTime = 0, cosTime = 0, sqrtTime = 0, t = 0;

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

timeT = clock();

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

sin((double)rand() / RAND\_MAX);

}

sinTime += clock() - timeT;

timeT = clock();

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

cos((double)rand() / RAND\_MAX);

}

cosTime += clock() - timeT;

timeT = clock();

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

sqrt((double)rand() / RAND\_MAX);

}

sqrtTime += clock() - timeT;

t += sinTime / CLOCKS\_PER\_SEC + cosTime / CLOCKS\_PER\_SEC + sqrtTime / CLOCKS\_PER\_SEC;

m += t / N;

m2 += pow(t, 2) / N;

}

dis = (double)m2 - pow(m, 2);

sinTime = sinTime / N / CLOCKS\_PER\_SEC;

cosTime = cosTime / N / CLOCKS\_PER\_SEC;

sqrtTime = sqrtTime / N / CLOCKS\_PER\_SEC;

out << "PModel: " << processor << endl;

out << "Task: " << "cos, sin, sqrt" << endl;

out << "OpType: " << "Double" << endl;

out << "Opt: " << key << endl;

out << "LNum: " << "sin: " << countSin << ", cos: " << countCos << ", sqrt: " << countSqrt << endl;

out << "InsCount: " << "sin: 1, cos: 1, sqrt: 1" << endl;

out << "AvTime: " << "sin: " << fixed << setprecision(9) << sinTime / countSin << ", cos: " << cosTime / countCos << ", sqrt: " << sqrtTime / countSqrt << endl;

out << "AbsError: " << fixed << setprecision(4) << sqrt(dis) << endl;

out << "RelError: " << sqrt(dis) / m<< "%" << endl;

out << "TaskPerf: " << (double)1 / (((double)countSin / sum / (countSin / sinTime)) + ((double)countCos / sum / (countCos / cosTime)) + ((double)countSqrt / sum / (countSqrt / sqrtTime)));

}

out.close();

}