Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Отчет по лабораторной работе

по дисциплине «Архитектура вычислительных систем»

Лабораторная работа №2

«Оценка производительности процессора»

Выполнил: студент 3 курса группы ИП-811 Мироненко К. А

Проверил: доцент кафедры ВС Ефимов А. В.

Оглавление

[1. Постановка задачи 3](#_Toc62078511)

[2. Выполнение работы 4](#_Toc62078512)

[3. Примеры работы программы 5](#_Toc62078513)

[*Приложение* Листинг 7](#_Toc62078514)

# Постановка задачи

***Тема:*** оценка производительности процессора

***Задание: р****еализовать 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.
2. С помощью системного таймера 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.
3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV.
4. **\*** Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).
5. **\*\*** Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).
6. **\*\*\*** Оценить и постараться минимизировать накладные расходы(время на вызов функций, влияние загрузки системы и т.п.) при испытании, то есть добиться максимальной точности измерений.
7. 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 целевого языка для написания тестов производительности был выбран язык C++. В роли типовых задач были выбрана функция перемножения матриц.

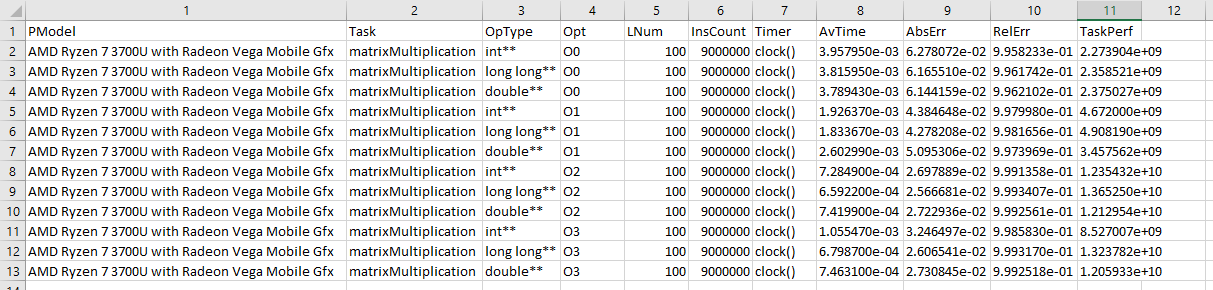
За само тестирование отвечали 3 функции void DGEMM(int\*\*, int\*\*, int\*\*, long long), void DGEMM(long long\*\*, long long\*\*, long long\*\*, long long), void DGEMM(double\*\*, double\*\*, double\*\*, long long), для каждого из типов данных соответственно.

Для получения сведений о процессоре был использован файл /proc/cpuinf.

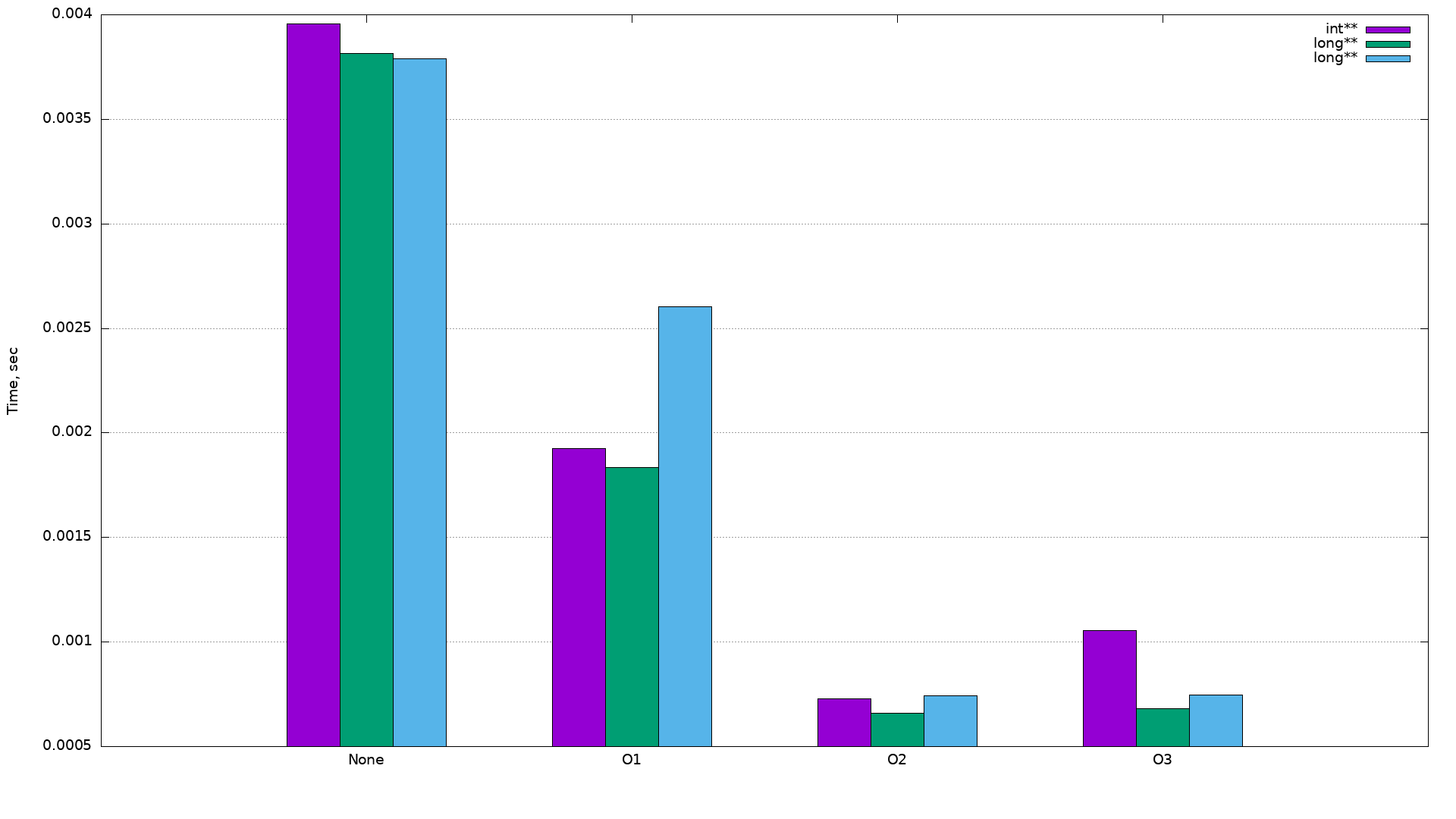
Затем все необходимые данные были сохранены в csv файл.

Чтобы оценить среднее время испытания каждой типовой задачи с разным типом входных данных и ключами оптимизации программа выполнялась несколько раз.

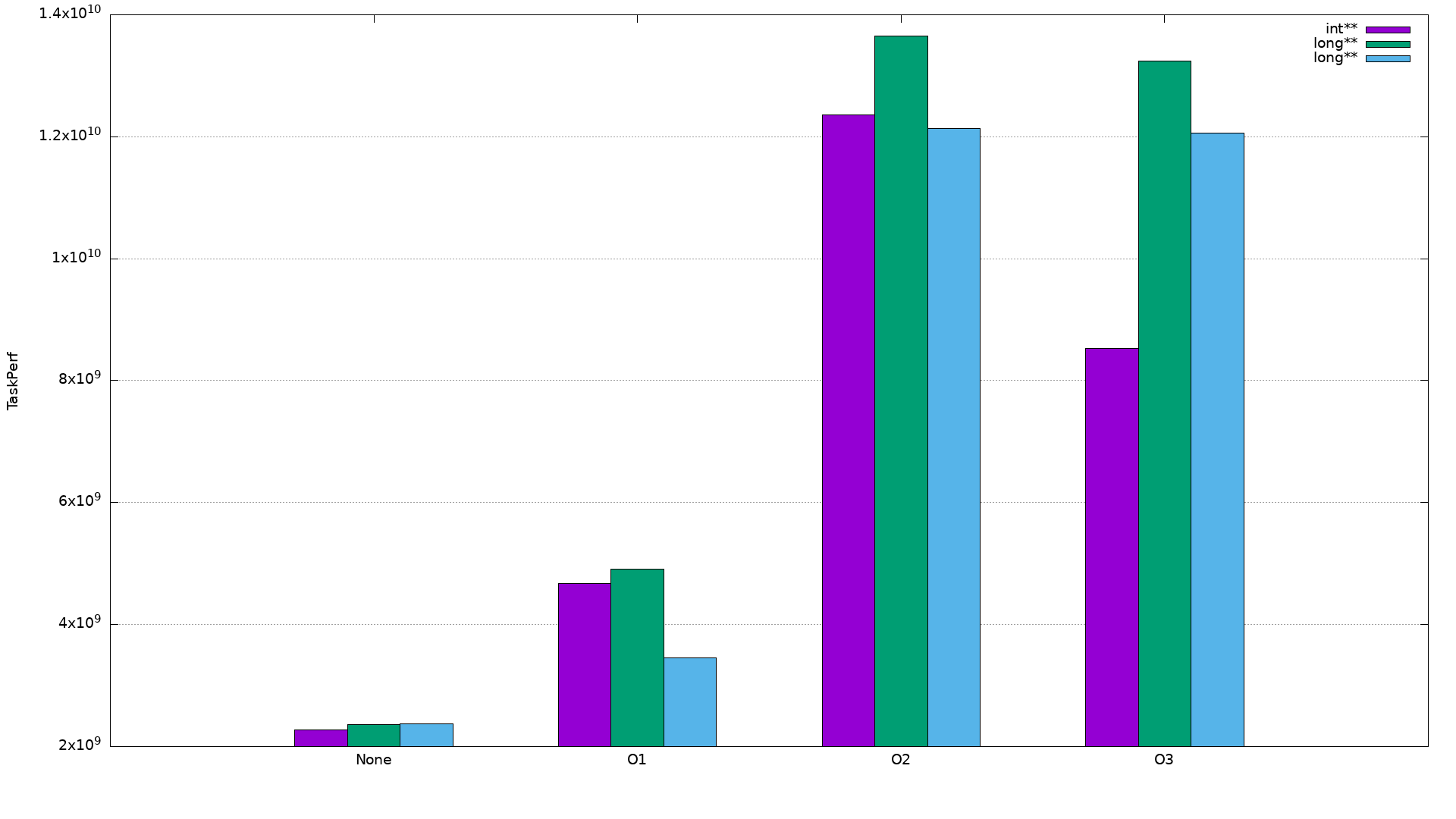
# Примеры работы программы



(csv таблица после выполнения программы)



(Гистограмма зависимости времени исполнения от степени оптимизации)



(Гистограмма производительности от степени оптимизации)

# *Приложение* Листинг

**main.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

#include <string.h>

#include <stdint.h>

#include <math.h>

void getNameCPU(char cpuname[]) {

FILE \*fp;

if ((fp = fopen("/proc/cpuinfo", "r")) == NULL) {

printf("Can't open /proc/cpuinfo \n");

return;

}

size\_t m = 0;

char \*line = NULL;

while (getline(&line, &m, fp) > 0) {

if (strstr(line, "model name")) {

strcpy(cpuname, &line[13]);

break;

}

}

for (int i = 0; i < 60; i++)

if (cpuname[i] == '\n'){

cpuname[i] = '\0';

break;

}

fclose(fp);

}

int getParameters(int argc, char \*argv[], long long &launchCount)

{

for (int i = 1 ; i < argc ; i++)

{

if (strcmp("-l", argv[i]) == 0 || strcmp("--launch-count", argv[i]) == 0)

{

i++;

size\_t len = strlen(argv[i]);

for (size\_t j = 0 ; j < len ; j++)

if (!isdigit(argv[i][j])){

printf("Error in arguments: invalid value for --launch-count!\nThe value must be a number!\n");

return 1;

}

launchCount = atoll(argv[i]);

}

else {

printf("Error in arguments: unknown key \"%s\"\n", argv[i]);

return 1;

}

}

return 0;

}

void DGEMM(int\*\* matrixA, int\*\* matrixB, int\*\* matrixC, long long matrixSize){

for (long long i = 0; i < matrixSize; i++)

for (long long j = 0; j < matrixSize; j++){

matrixC[i][j] += 0;

for (long long k = 0; k < matrixSize; k++)

matrixC[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

void DGEMM(long long\*\* matrixA, long long\*\* matrixB, long long\*\* matrixC, long long matrixSize){

for (long long i = 0; i < matrixSize; i++)

for (long long j = 0; j < matrixSize; j++){

matrixC[i][j] += 0;

for (long long k = 0; k < matrixSize; k++)

matrixC[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

void DGEMM(double\*\* matrixA, double\*\* matrixB, double\*\* matrixC, long long matrixSize){

for (long long i = 0; i < matrixSize; i++)

for (long long j = 0; j < matrixSize; j++){

matrixC[i][j] += 0;

for (long long k = 0; k < matrixSize; k++)

matrixC[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

int outToCSV(char\* nameCPU, char\* type, long long launchCount, double averageTime, double\* time, double dispersion, long long matrixSize)

{

FILE \*fp;

if (!(fp = fopen("output.csv", "a"))){

printf("Error: can't open/find output.csv\n");

return 1;

}

fprintf(fp, "%s;%s;%s;%s;%lld;%lld;%s;%e;%e;%e;%e;\n",

nameCPU,

"matrixMultiplication",

type,

"O0",

launchCount,

launchCount \* matrixSize \* matrixSize,

"clock()",

averageTime,

sqrt(abs(dispersion)),

abs(dispersion) / averageTime,

(launchCount \* matrixSize \* matrixSize) / averageTime);

fclose(fp);

return 0;

}

int main(int argc, char \*argv[]) {

srand(time(0));

long long matrixSize = 100;

long long launchCount = 100;

char cpuname[60];

getNameCPU(cpuname);

clock\_t start, stop;

if (getParameters(argc, argv, launchCount))

return 1;

printf("matrixSize = %lld\n", matrixSize);

printf("launchCount = %lld\n", launchCount);

double timeSum = 0;

double dispersion = 0;

double summand1 = 0;

double summand2 = 0;

double time[launchCount];

//////////////////////////

timeSum = 0;

summand1 = 0;

summand2 = 0;

int \*\*matrixA\_i = new int\*[matrixSize];

int \*\*matrixB\_i = new int\*[matrixSize];

int \*\*matrixRes\_i = new int\*[matrixSize];

for (long long i = 0; i < matrixSize; i++) {

matrixA\_i[i] = new int[matrixSize];

matrixB\_i[i] = new int[matrixSize];

matrixRes\_i[i] = new int[matrixSize];

for (long long j = 0; j < matrixSize; j++) {

matrixA\_i[i][j] = rand() / 10000;

matrixB\_i[i][j] = rand() / 10000;

matrixRes\_i[i][j] = 0;

}

}

for (long long i = 0; i < launchCount; i++) {

start = clock();

DGEMM(matrixA\_i, matrixB\_i, matrixRes\_i, matrixSize);

stop = clock();

time[i] = ((double)(stop - start)) / CLOCKS\_PER\_SEC;

timeSum += time[i];

summand1 += time[i] \* time[i];

summand2 += time[i];

}

dispersion = summand1 / launchCount - summand2 / launchCount;

outToCSV(cpuname, (char\*)"int\*\*", launchCount, timeSum / launchCount, time, dispersion, matrixSize);

for (long long i = 0; i < matrixSize; i++) {

delete(matrixA\_i[i]);

delete(matrixB\_i[i]);

delete(matrixRes\_i[i]);

}

delete[](matrixA\_i);

delete[](matrixB\_i);

delete[](matrixRes\_i);

//////////////////////////

timeSum = 0;

summand1 = 0;

summand2 = 0;

long long \*\*matrixA\_ll = new long long\*[matrixSize];

long long \*\*matrixB\_ll = new long long\*[matrixSize];

long long \*\*matrixRes\_ll = new long long\*[matrixSize];

for (long long i = 0; i < matrixSize; i++) {

matrixA\_ll[i] = new long long[matrixSize];

matrixB\_ll[i] = new long long[matrixSize];

matrixRes\_ll[i] = new long long[matrixSize];

for (long long j = 0; j < matrixSize; j++) {

matrixA\_ll[i][j] = rand() / 10000;

matrixB\_ll[i][j] = rand() / 10000;

matrixRes\_ll[i][j] = 0;

}

}

for (long long i = 0; i < launchCount; i++) {

start = clock();

DGEMM(matrixA\_ll, matrixB\_ll, matrixRes\_ll, matrixSize);

stop = clock();

time[i] = ((double)(stop - start)) / CLOCKS\_PER\_SEC;

timeSum += time[i];

summand1 += time[i] \* time[i];

summand2 += time[i];

}

dispersion = summand1 / launchCount - summand2 / launchCount;

outToCSV(cpuname, (char\*)"long long\*\*", launchCount, timeSum / launchCount, time, dispersion, matrixSize);

for (long long i = 0; i < matrixSize; i++) {

delete(matrixA\_ll[i]);

delete(matrixB\_ll[i]);

delete(matrixRes\_ll[i]);

}

delete[](matrixA\_ll);

delete[](matrixB\_ll);

delete[](matrixRes\_ll);

//////////////////////////

timeSum = 0;

summand1 = 0;

summand2 = 0;

double \*\*matrixA\_d = new double\*[matrixSize];

double \*\*matrixB\_d = new double\*[matrixSize];

double \*\*matrixRes\_d = new double\*[matrixSize];

for (long long i = 0; i < matrixSize; i++) {

matrixA\_d[i] = new double[matrixSize];

matrixB\_d[i] = new double[matrixSize];

matrixRes\_d[i] = new double[matrixSize];

for (long long j = 0; j < matrixSize; j++) {

matrixA\_d[i][j] = rand() / 10000 + (double)rand() / RAND\_MAX;

matrixB\_d[i][j] = rand() / 10000 + (double)rand() / RAND\_MAX;

matrixRes\_d[i][j] = 0;

}

}

for (long long i = 0; i < launchCount; i++) {

start = clock();

DGEMM(matrixA\_d, matrixB\_d, matrixRes\_d, matrixSize);

stop = clock();

time[i] = ((double)(stop - start)) / CLOCKS\_PER\_SEC;

timeSum += time[i];

summand1 += time[i] \* time[i];

summand2 += time[i];

}

dispersion = summand1 / launchCount - summand2 / launchCount;

outToCSV(cpuname, (char\*)"double\*\*", launchCount, timeSum / launchCount, time, dispersion, matrixSize);

for (long long i = 0; i < matrixSize; i++) {

delete(matrixA\_d[i]);

delete(matrixB\_d[i]);

delete(matrixRes\_d[i]);

}

delete[](matrixA\_d);

delete[](matrixB\_d);

delete[](matrixRes\_d);

return 0;

}