ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3  
«Оценка производительности подсистемы памяти»  
по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИА-832

Тиванов Данил Евгеньевич

Проверил: ст. преп. Кафедры ВС   
Ткачева Татьяна Алексеевна

Новосибирск 2020

Содержание

[Постановка задачи 3](#_Toc50517127)

[Выполнение работы](#_Toc50517128) 6

[Результат работы](#_Toc50517129) 7

[Приложение 1](#_Toc50517130)4

# Постановка задачи

1. Написать программу(функцию) на языке С/С++/C# для оценки производительности подсистемы памяти.

На вход программы подать следующие аргументы.

1) Подсистема памяти. Предусмотреть возможность указать подсистему для проверки производительности: RAM (оперативная память), HDD/SSD и flash.

2) Размер блока данных в байтах, Кб или Мб. Если размерность не указана, то в байтах, если указана, то соответственно в Кбайтах или Мбайтах.

3) Число испытаний, т.е. число раз повторений измерений.

Пример вызова программы: ./memory\_test –m RAM –b 1024|1Kb –l 10 или ./memory\_bandwidth

––memory-type RAM|HDD|SSD|flash

––block-size 1024|1Kb

––launch-count 10

В качестве блока данных использовать одномерный массив, в котором произведение числа элементов на их размерность равна требуемому размеру блока данных. Массив инициализировать случайными значениями. Для тестирования HDD/SSD и flash создать в программе файлы в соответствующих директориях.

Измерение времени реализовать с помощью функции clock\_gettime() или аналогичной с точность до наносекунд. Измерять время исключительно на запись элемента в память или считывание из неё, без операций генерации или преобразования данных.

На выходе программы в одну строку CSV файла со следующей структурой:

[MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidthAbsError(read);RelError(read);], где

MemoryType – тип памяти (RAM|HDD|SSD|flash) или модель устройства, на котором проводятся испытания;

BlockSize – размер блока данных для записи и чтения на каждом испытании;

ElementType – тип элементов используемых для заполнения массива данных;

BufferSize – размер буфера, т.е. порции данных для выполнения одно операции записи или чтения;

LaunchNum – порядковый номер испытания;

Timer – название функции обращения к таймеру (для измерения времени);

WriteTime – время выполнения отдельного испытания с номером LaunchNum [секунды];

AverageWriteTime – среднее время записи из LaunchNum испытаний [секунды];

WriteBandwidth – пропускная способность памяти (BLOCK\_SIZE/AverageWriteTime) / 10^6 [Mb/s]

AbsError(write) – абсолютная погрешность измерения времени записи или СКО [секунды];

RelError(write) – относительная погрешность измерения времени [%];

ReadTime – время выполнения отдельного испытания LaunchNum [секунды];

AverageReadTime – среднее время записи из LaunchNum испытаний [секунды];

ReadBandwidth – пропускная способность памяти (BLOCK\_SIZE/AverageReadTime) / 10^6[Mб/сек.]

AbsError(read) – абсолютная погрешность измерения времени чтения или СКО [секунды];

RelError(read) – относительная погрешность измерения времени [%].

2. Написать программу(функцию) на языке С/С++/C# или скрипт (benchmark) реализующий серию испытаний программы(функции) из п.1. Оценить пропускную способность оперативной памяти при работе с блоками данных равными объёму кэш-линии, кэш-памяти L1, L2 и L3 уровня и превышающего его. Для HDD|SSD и flash провести серию из 20 испытаний с блоками данных начиная с 4 Мб с шагом 4Мб. Результаты всех испытаний сохранить в один CSV файл со структурой, описанной в п.1.

3. На основе CSV файла построить сводные таблицы и диаграммы отражающие:

1) Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти;

2) Зависимость погрешности измерения пропускной способности от размера блока данных для разного типа памяти;

3) Зависимость погрешности измерений от числа испытаний LaunchNum;

# Выполнение работы

Для тестирования оперативной памяти используется массив типа int. Так как int кодируется 4 байтами, то размер массива будет в 4 раза меньше. Для тестирования SSD, HDD и flash создается файл, куда записываются и считываются данные (в данном случае типа int).

Для получения случайных значений и заполнения массива/записи в файл используется функция rand().

Для измерения времени в наносекундах используется библиотека времени chrono, пользволяющая делать измерения в наносекундах. Функция chrono::steady\_clock::now()возвращает текущий момент времени.

Для записи в файл используется библиотека fstream. С помощью функции peek() производится проверка записана ли шапка таблицы в файл. Если нет, то она записывается, иначе пропускается строка и записываются вычисленные данные.

Для вычисления пропускной способности при записи используется формула (BLOCK\_SIZE/AverageWriteTime(AverageReadTime)) / 10^6[Mb/s], где BLOCK\_SIZE - размер блока данных для записи и чтения, AverageWriteTime (AverageReadTime) - среднее время записи(чтения) из LaunchNum испытаний [секунды].

# Результат работы

Результат работы программы с блоками данных от 4 Мб с шагом 4 Мб c HDD представлен на рисунке 1.

Рис.1 - результат работы программы с 20 испытаниями HDD с блоком данных от 4 Мб с шагом 4 Мб

Результат работы программы с блоками данных от 4 Мб с шагом 4 Мб c flash представлен на рисунке 2.

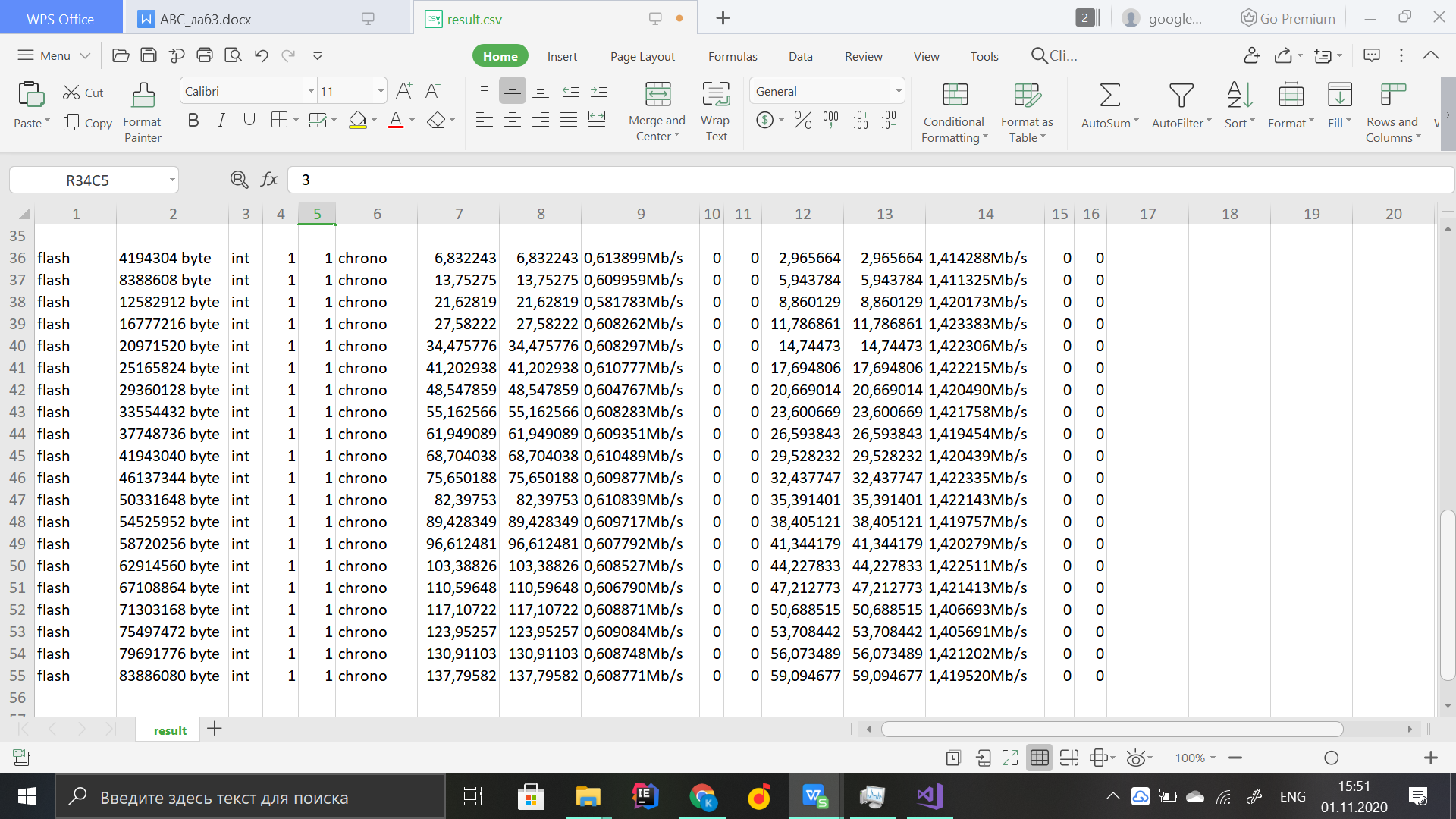


Рис.2 - результат работы программы с 20 испытаниями flash с блоком данных от 4 Мб с шагом 4 Мб

Результат работы программы с блоками данных, равными объему кэш памяти и выше, представлен на рисунке 3.

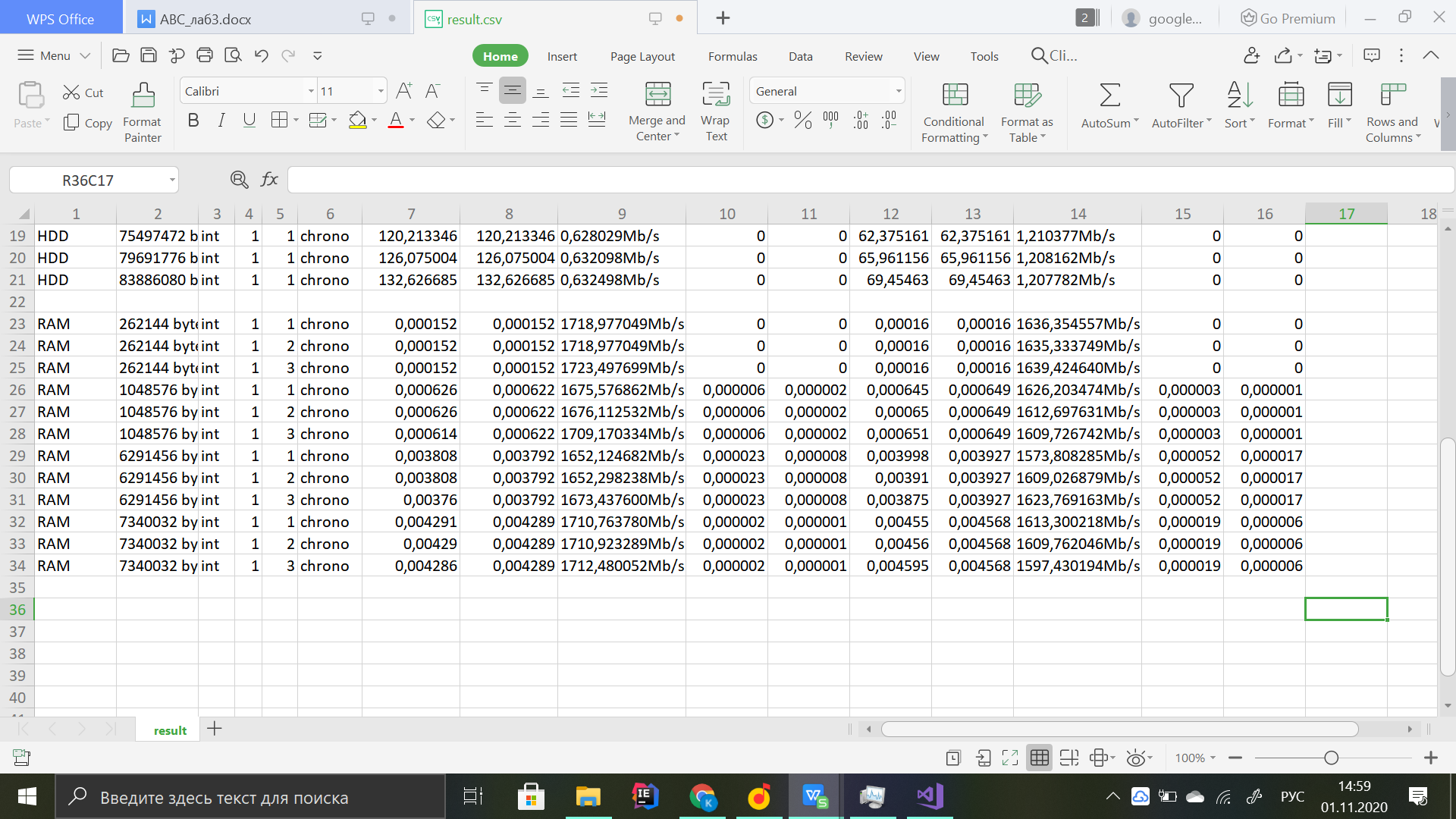


Рис.3 - результат работы программы с блоками данных, равными размеру кэш памяти (262144, 1048576 и 6291456 байт соответственно) и выше (7340032 байт), с 3 повторениями

Результат работы программы с SSD с размером 1024 Kb и 5 повторениями представлен на рисунке 4.

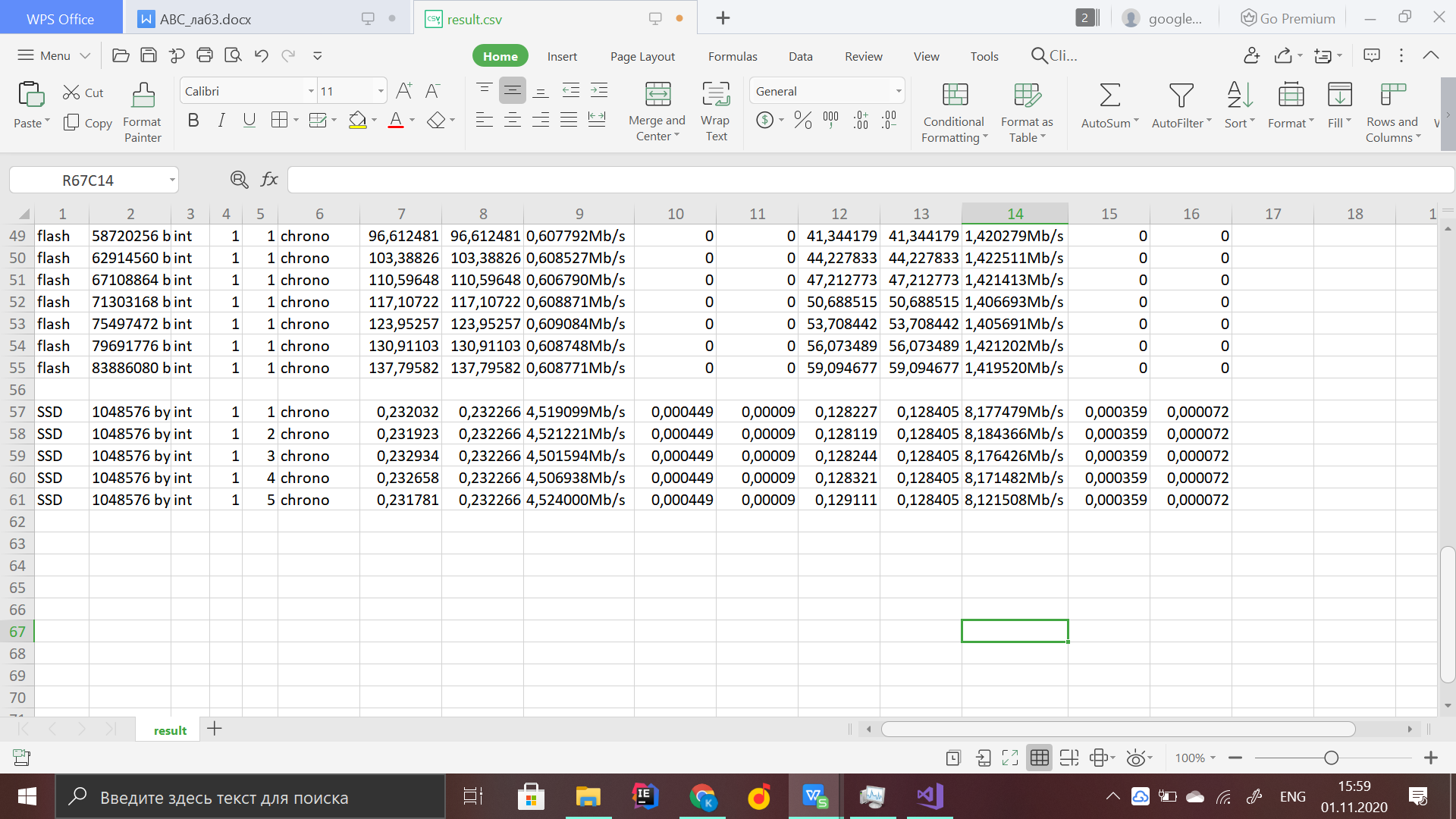


Рис.4 - результат работы программы при ручном вводе с SSD с размером 1024 Kb и 5 повторениями

На основании полученных данных были построены сводные диаграммы зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти, диаграммы зависимости погрешности измерения пропускной способности от размера блока данных для разного типа памяти, диаграммы зависимости погрешности измерений от числа испытаний LaunchNum.

Диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для типа памяти HDD представлена на рисунке 5.

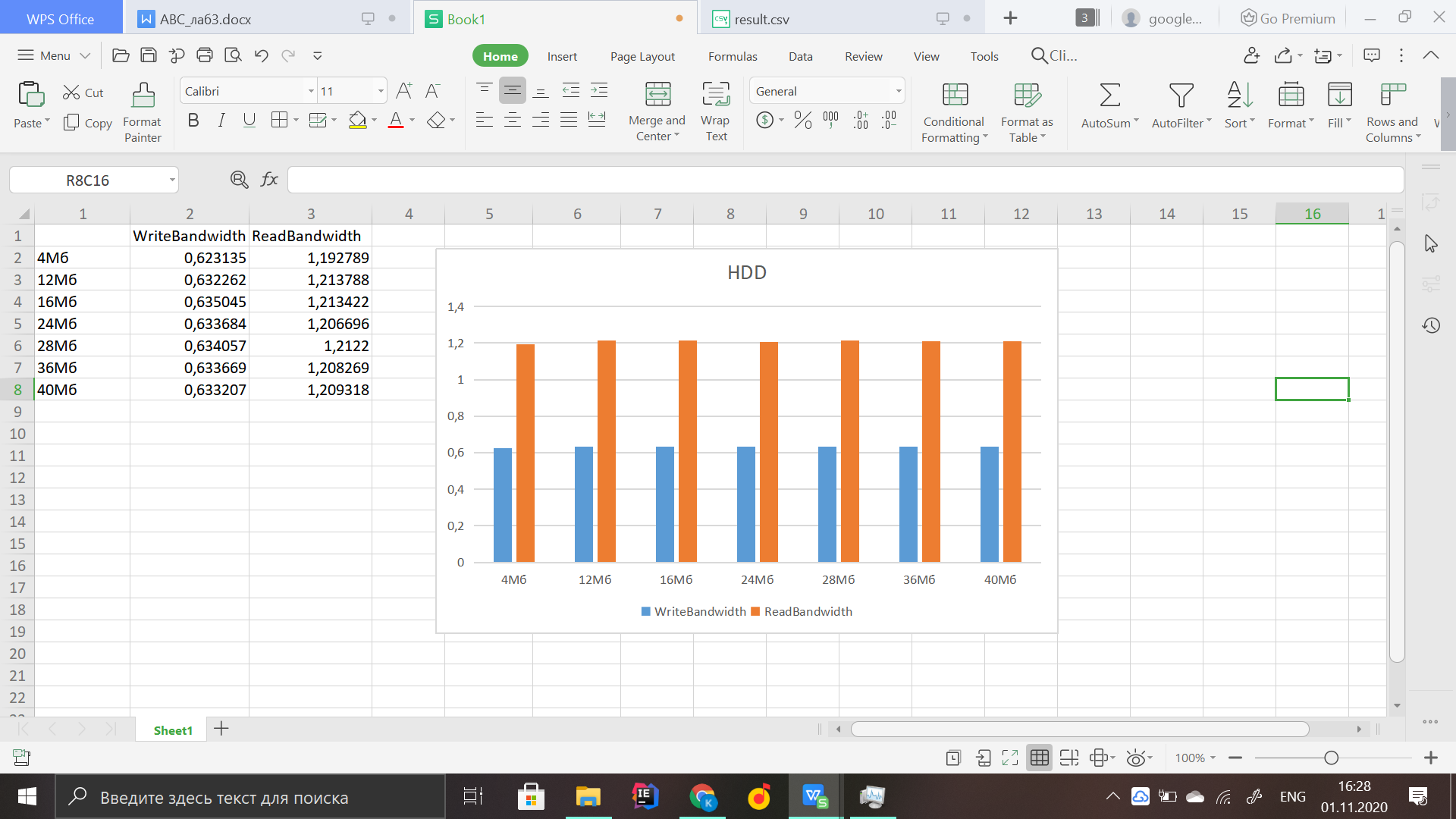


Рис.5 - диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize равных 4 Мб, 12 Мб, 16 Мб, 24 Мб, 28 Мб, 36 Мб, 40 Мб) для HDD

Диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для типа памяти SSD представлена на рисунке 6.

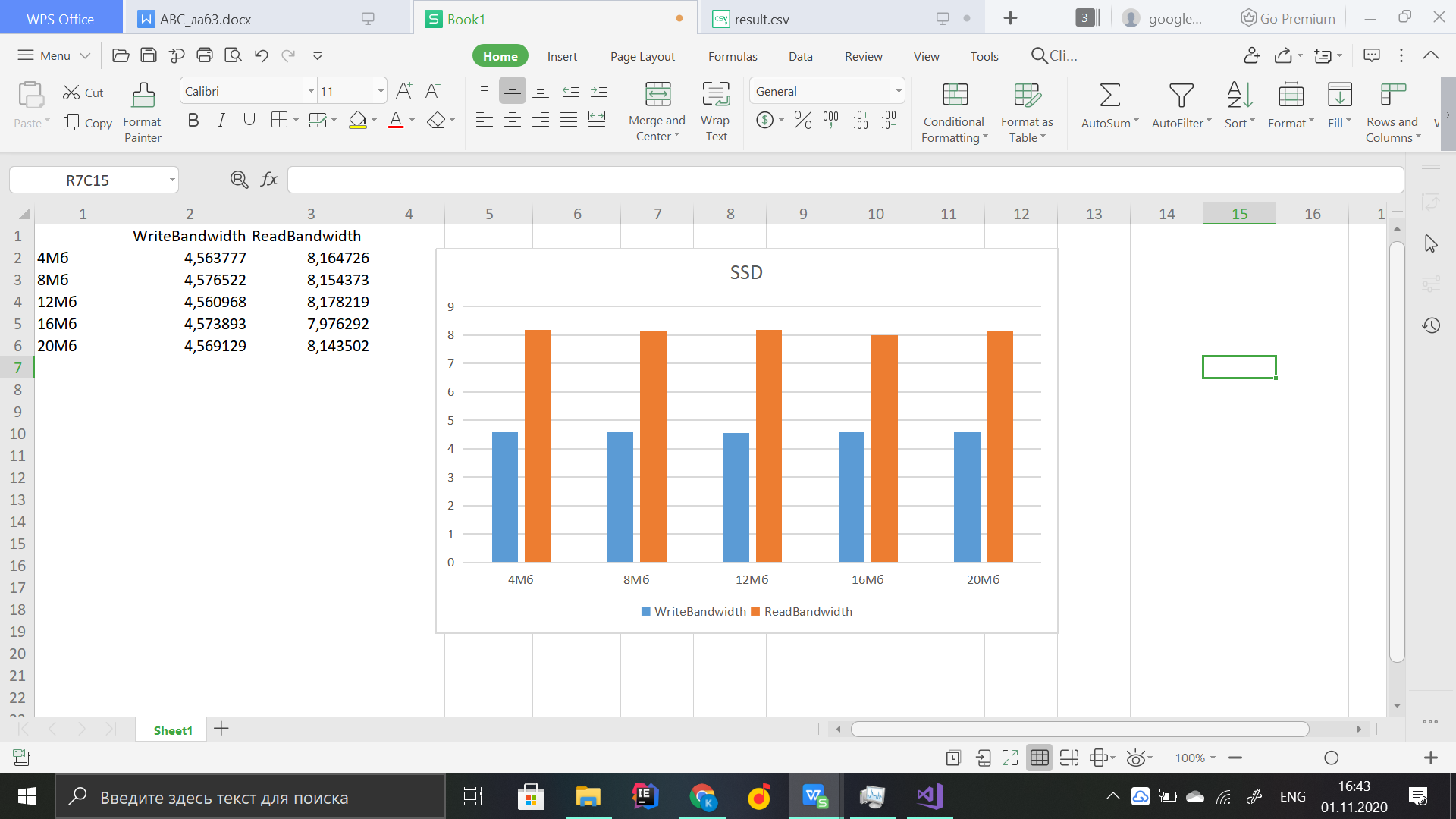


Рис.6 - диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize равных 4 Мб, 8 Мб, 12 Мб, 16 Мб, 20 Мб) для SSD

Диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для типа памяти flash представлена на рисунке 7.

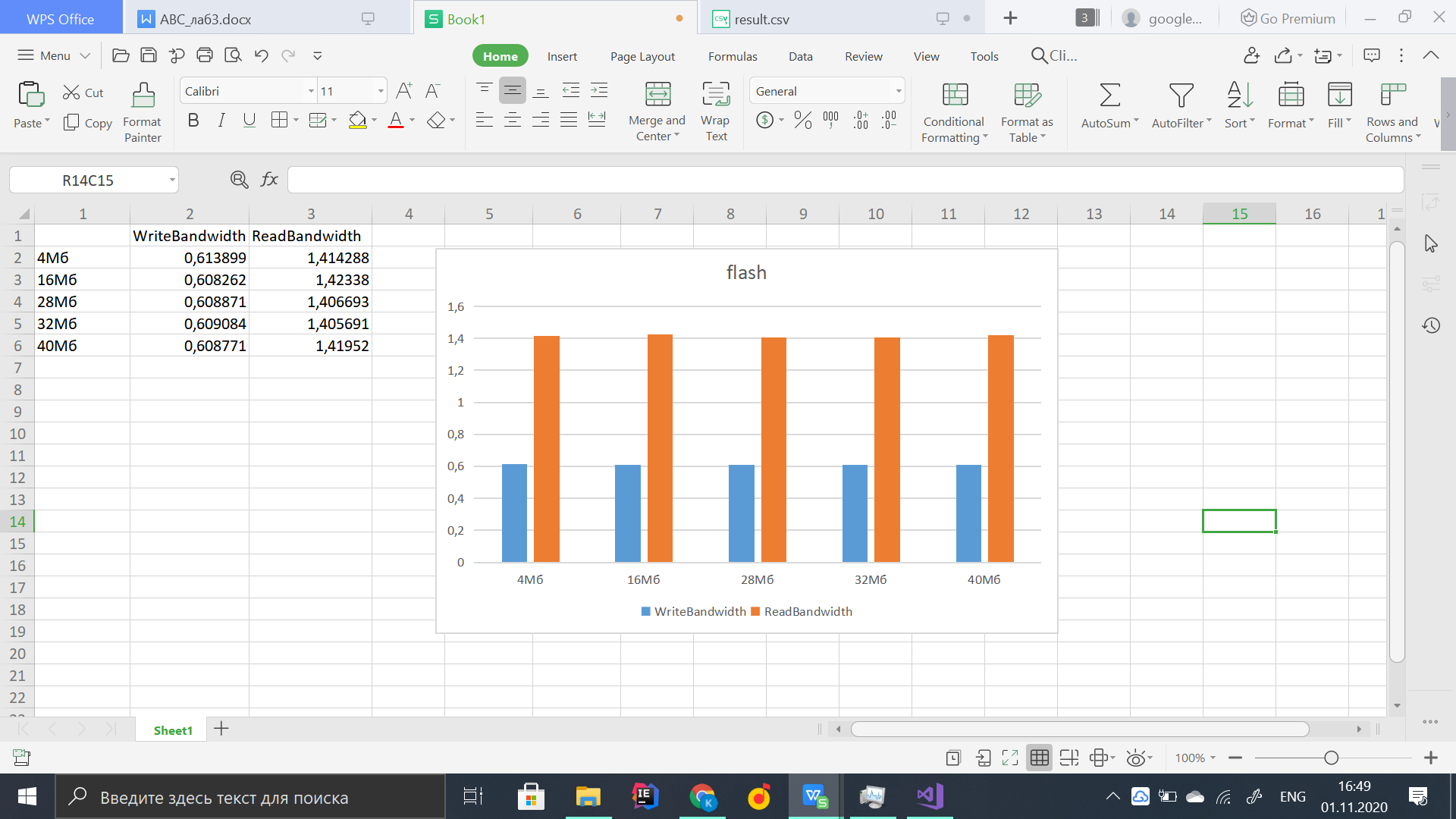


Рис.7 - диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize равных 4 Мб, 16 Мб, 28 Мб, 32 Мб, 40 Мб) для flash

Диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize) для типа памяти RAM представлена на рисунке 8.

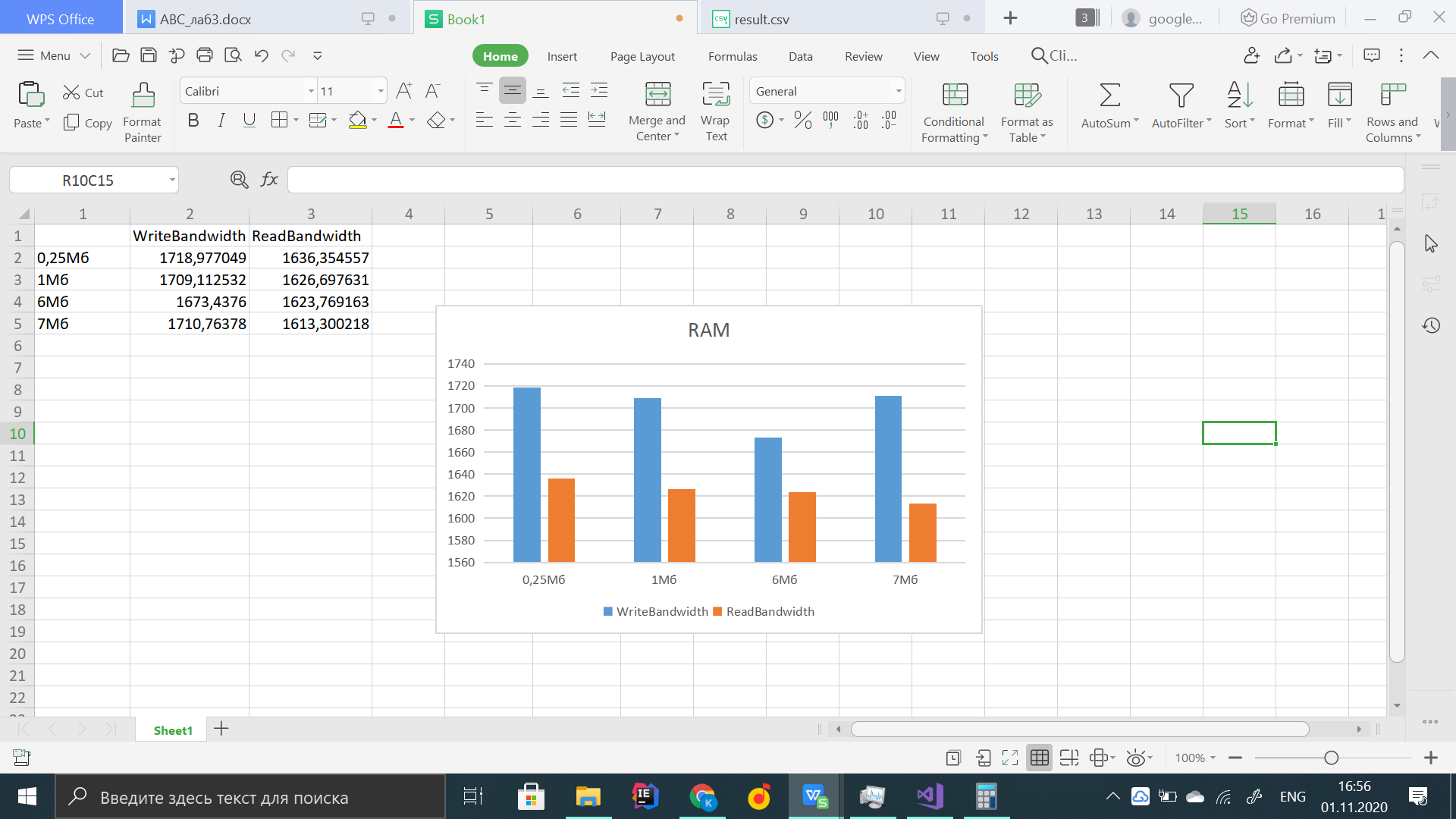


Рис.8 - диаграмма зависимости пропускной способности записи и чтения от размера блока данных (BlockSize равных 256 Кб, 1 Мб, 6 Мб, 7 Мб) для RAM

Из полученных диаграмм можно сделать вывод, что скорость записи(чтения) не зависит от размера блока данных.

Диаграмма зависимости погрешности измерения пропускной способности от размера блока данных для типа памяти SSD представлена на рисунке 9.

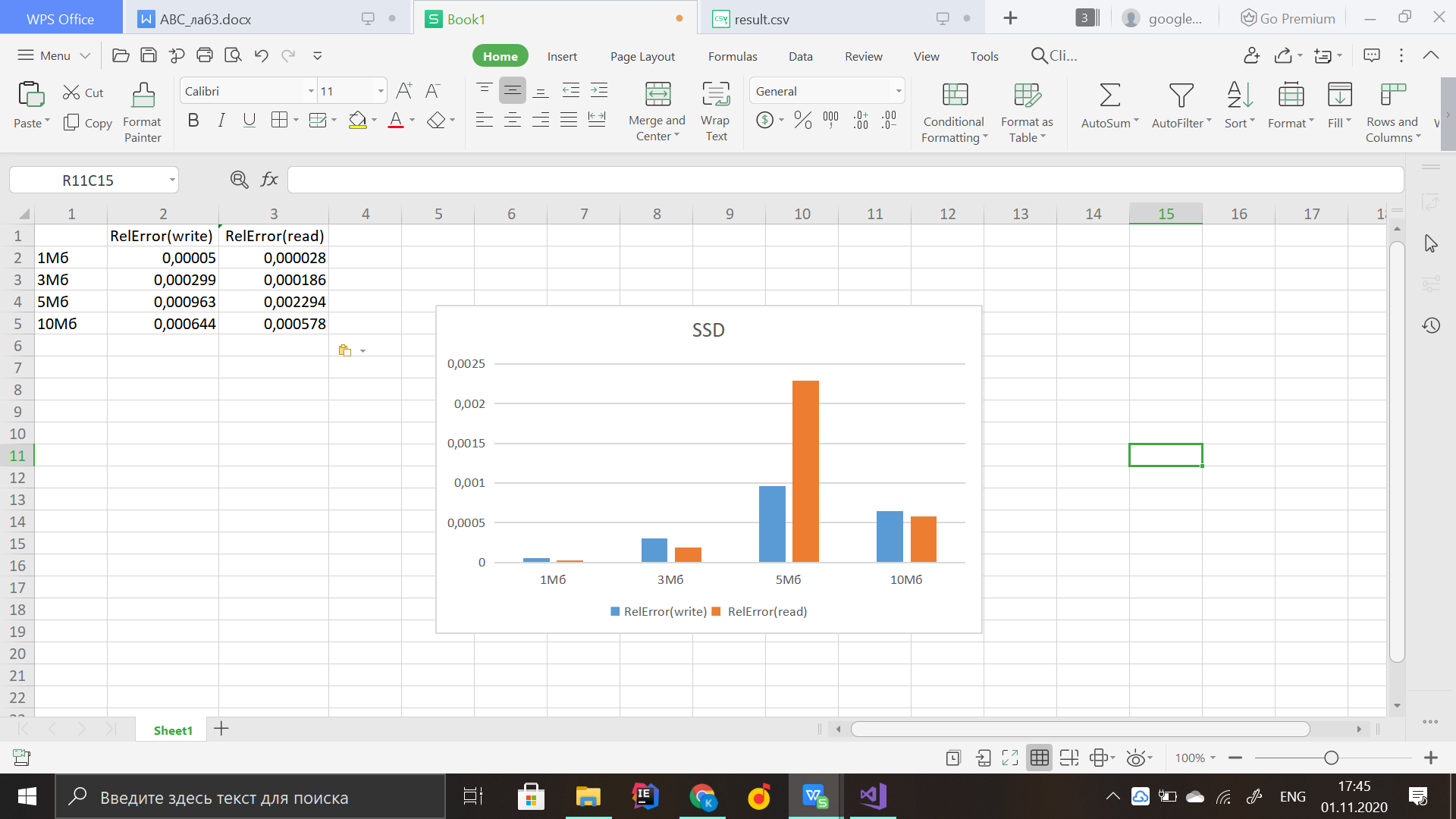


Рис.9 - диаграмма зависимости погрешности измерения пропускной способности от размера блока данных (1 Мб, 3 Мб, 5 Мб, 10 Мб) для типа памяти SSD

Диаграмма зависимости погрешности измерения пропускной способности от размера блока данных для типа памяти HDD представлена на рисунке 10.

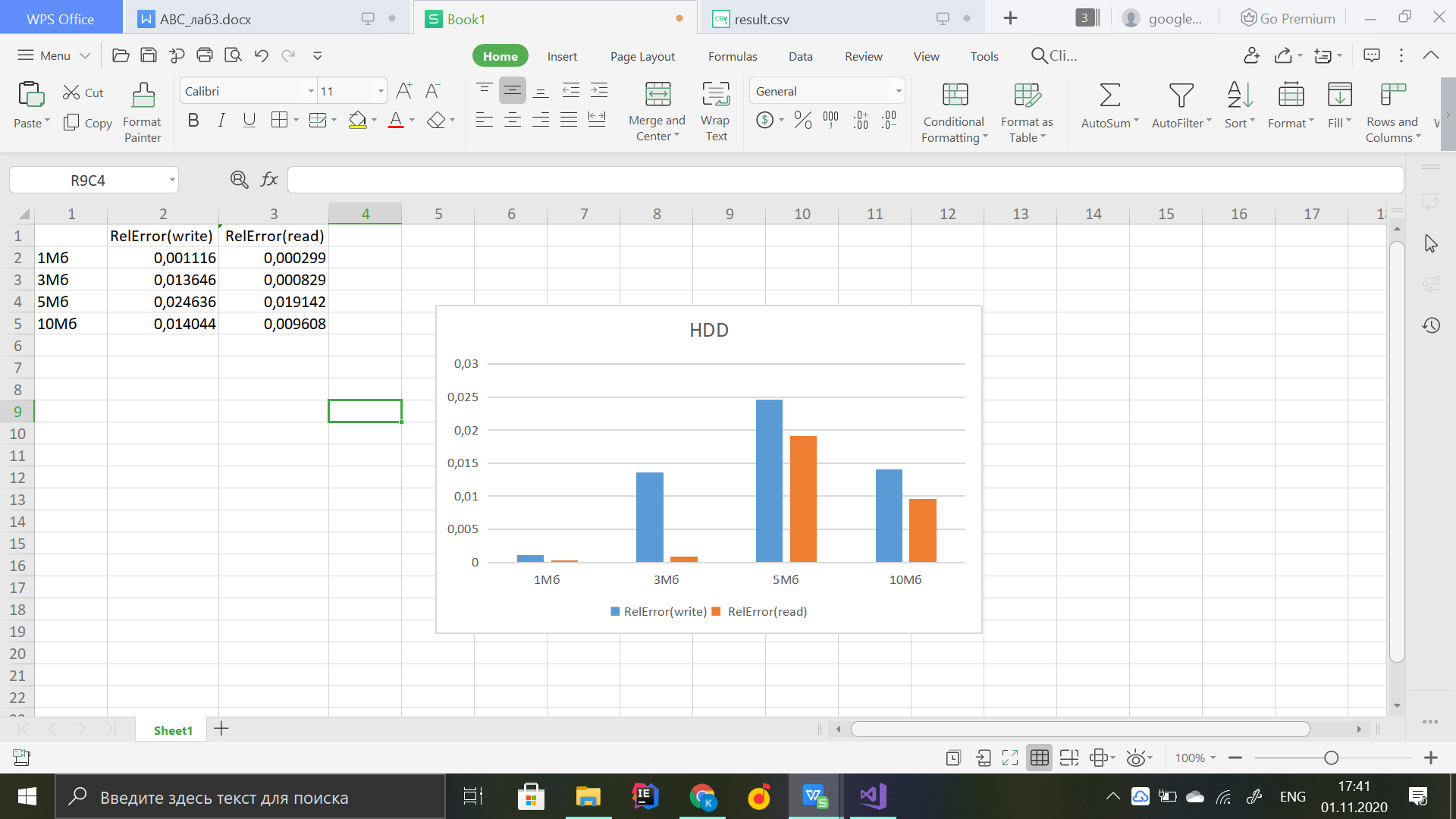


Рис.10 - диаграмма зависимости погрешности измерения пропускной способности от размера блока данных (1 Мб, 3 Мб, 5 Мб, 10 Мб) для типа памяти HDD

Диаграмма зависимости погрешности измерения пропускной способности от размера блока данных для типа памяти flash представлена на рисунке 11.

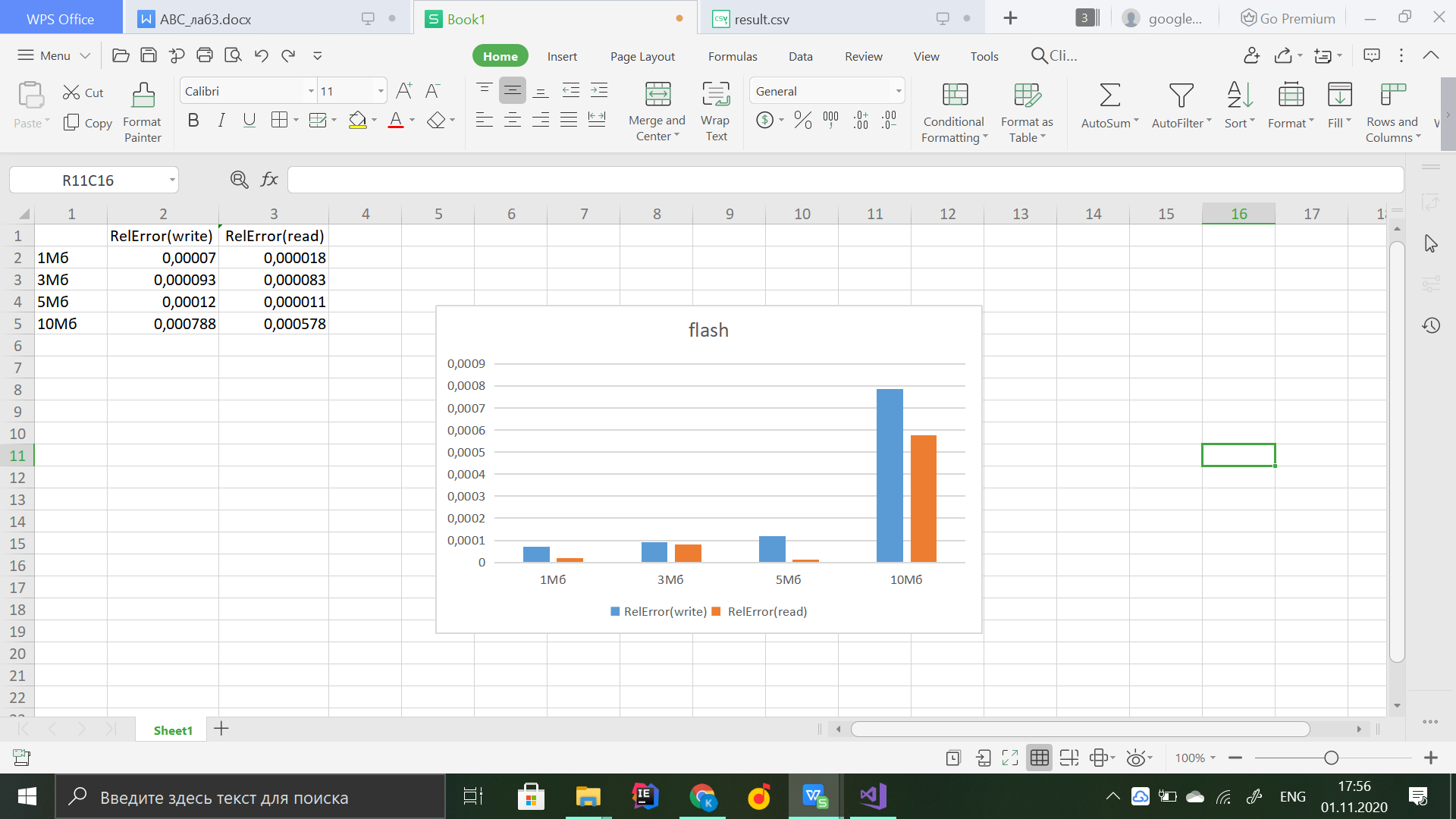


Рис.11 - диаграмма зависимости погрешности измерения пропускной способности от размера блока данных (1 Мб, 3 Мб, 5 Мб, 10 Мб) для типа памяти flash

Диаграмма зависимости погрешности измерения пропускной способности от размера блока данных для типа памяти RAM представлена на рисунке 12.

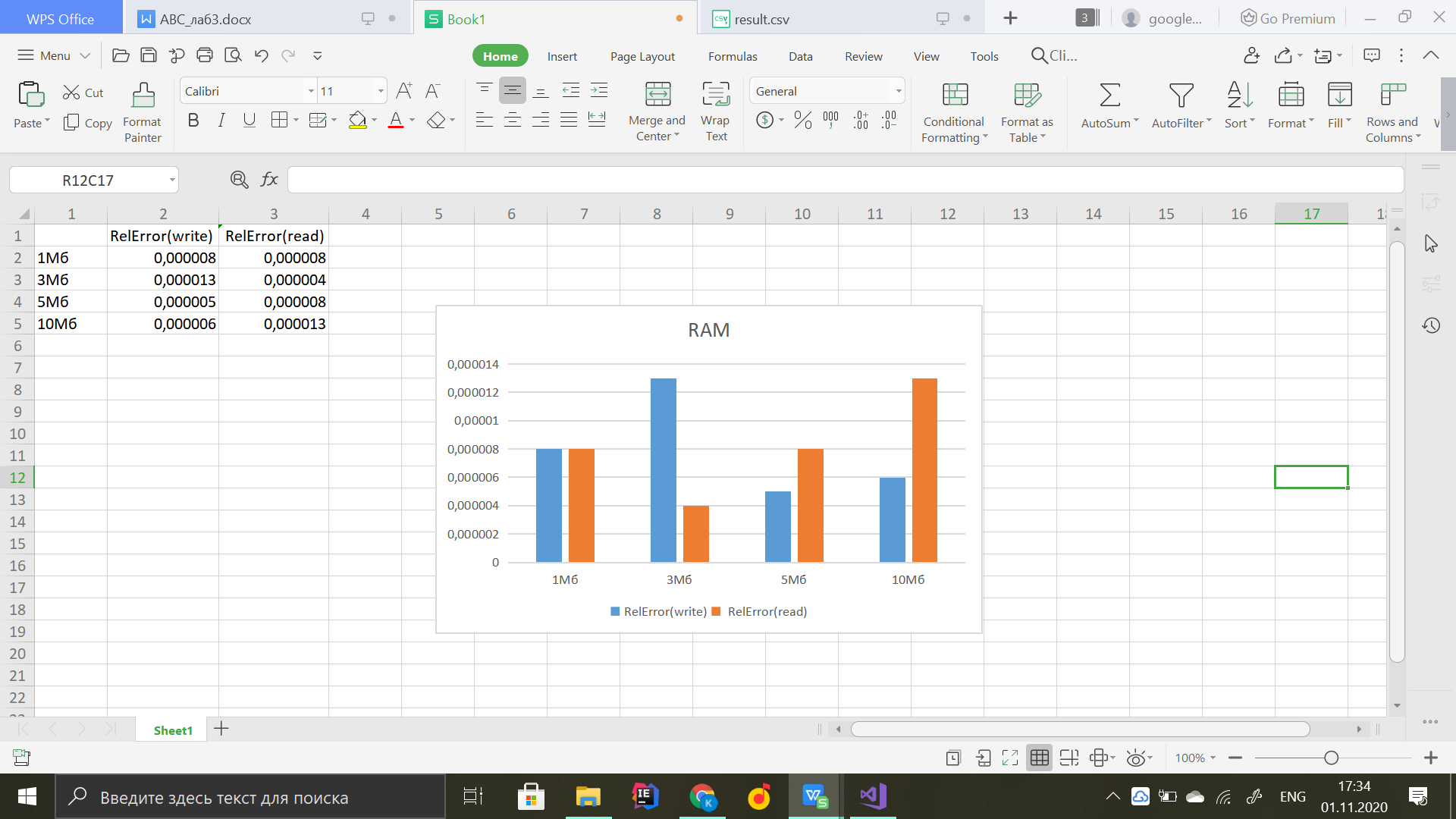


Рис.12 - диаграмма зависимости погрешности измерения пропускной способности от размера блока данных (1 Мб, 3 Мб, 5 Мб, 10 Мб) для типа памяти RAM

Диаграмма зависимости погрешности измерений от числа испытаний LaunchNum представлена на рисунке 13.

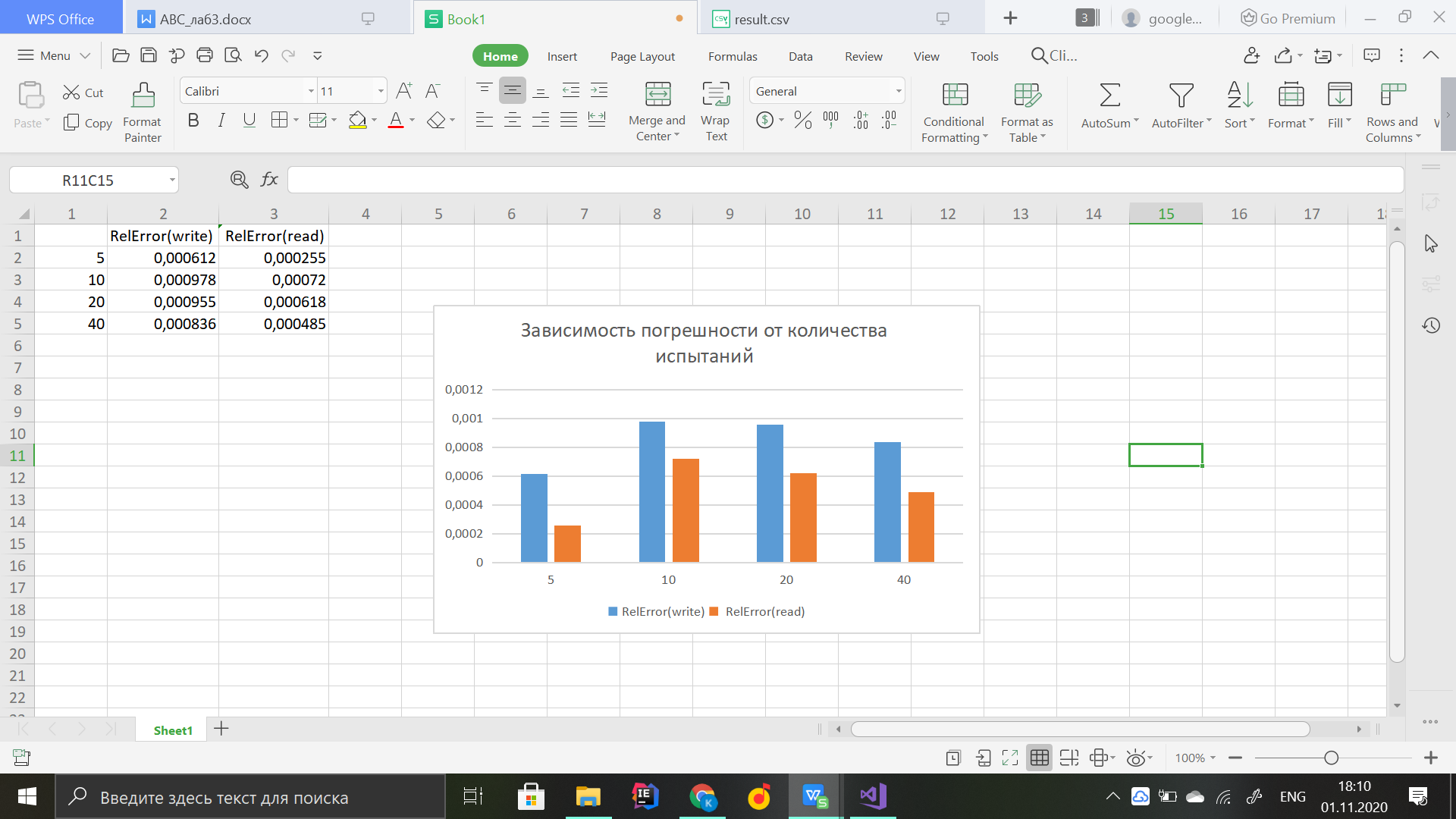


Рис.13 - диаграмма зависимости погрешности измерений от числа испытаний LaunchNum (5, 10, 20, 40)

# Приложение

#include <iostream>

#include <string>

#include <chrono>

#include <fstream>

using namespace std;

void testRAM(long size, long double &timeOfWrite, long double &timeOfRead) {

long long int length = size / 4;

int\* arr = new int[length];

int\* temp = new int[length];

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

temp[i] = rand();

}

auto start = chrono::steady\_clock::now();

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

arr[i] = temp[i];

}

auto end = chrono::steady\_clock::now();

chrono::duration<double> elapsed\_seconds = end - start;

timeOfWrite = elapsed\_seconds.count();

delete[] temp;

char t;

start = chrono::steady\_clock::now();

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

t = arr[i];

}

end = chrono::steady\_clock::now();

elapsed\_seconds = end - start;

timeOfRead = elapsed\_seconds.count();

delete[] arr;

}

void testMemory(string address, long size, long double &timeOfWrite, long double &timeOfRead) {

ofstream out(address + "\testMemory.txt");

int len = size / 2;

int \*s = new int[len];

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

s[i] = (rand() \* ( 1.0 / (static\_cast<double>(RAND\_MAX) + 1.0)) \* 10);

}

auto start = chrono::steady\_clock::now();

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

out << s[i] << " ";

}

auto end = chrono::steady\_clock::now();

chrono::duration<double> elapsed\_seconds = end - start;

out.close();

timeOfWrite = elapsed\_seconds.count();

ifstream in(address + "\testMemory.txt");

string now;

start = chrono::steady\_clock::now();

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

in >> now;

}

end = chrono::steady\_clock::now();

elapsed\_seconds = end - start;

in.close();

timeOfRead = elapsed\_seconds.count();

delete s;

}

void printToFile(string type, string sizeType, int count, long size, long double rdis, long double\* read,

long double\* write, long double srread, long double srwrite, long double wrdis, int buffer) {

fstream fileo("result.csv", ios\_base::app);

if (ifstream("result.csv").peek() == EOF) {

fileo << "MemoryType;" << "BlockSize;" << "ElementType;" << "BufferSize;" << "LaunchNum;" << "Timer;" << "WriteTime;" << "AverageWriteTime;" <<

"WriteBandwidth;" << "AbsError(write);" << "RelError(write);" << "ReadTime;" << "AverageReadTime;" << "ReadBandwidth;" <<

"AbsError(read); " << "RelError(read);" << endl;

}

else {

fileo << endl;

}

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

fileo << type << ";" << size << " byte;" << "int" <<";" << buffer << ";" << (i + 1) << ";" << "chrono;" << to\_string(write[i]) << ";" << to\_string(srwrite) << ";"

<< to\_string(size / write[i]/ 1000000) << "Mb/s;" << to\_string(sqrt(wrdis)) << ";" << to\_string(sqrt(wrdis) / count) << ";" << to\_string(read[i]) << ";"

<< to\_string(srread) << ";" << to\_string(size / read[i] / 1000000) << "Mb/s;" << to\_string(sqrt(rdis)) << ";" << to\_string(sqrt(rdis) / count) << ";" << endl;

}

fileo.close();

}

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

string type, sizeType, address = "";

int count = 3;

int userInput;

long l[] = {262144, 1048576, 6291456, 7340032};

long size = 0;

string memoryType[] = {"HDD", "flash"};

long double wrm2 = 0, wrm = 0, wrdis, rm = 0, rm2 = 0, rdis = 0, srwrite = 0, srread = 0;

int num, numStep;

cout << "1 - ручной ввод" << endl << "2 - тест оперативной памяти" << endl << "3 - тест из 20 испытаний" << endl;

cin >> userInput;

switch (userInput)

{

case 1:

cout << "Пример входных данных: RAM 1024 Kb 10\n";

cin >> type;

cin >> size;

cin >> sizeType;

cin >> count;

if (sizeType == "Mb") {

size \*= 1048576;

}

else if (sizeType == "Kb") {

size \*= 1024;

}

num = 1;

numStep = 1;

break;

case 2:

type = "RAM";

num = 4;

break;

case 3:

num = 2;

count = 1;

numStep = 20;

break;

}

long double\* write = new long double[count];

long double\* read = new long double[count];

if (type == "RAM") {

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

if (num > 1) {

size = l[j];

wrm2 = 0, wrm = 0, wrdis, rm = 0, rm2 = 0, rdis = 0, srwrite = 0, srread = 0;

}

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

testRAM(size, write[i], read[i]);

wrm += write[i] / count;

wrm2 += pow(write[i], 2) / count;

rm += read[i] / count;

rm2 += pow(read[i], 2) / count;

srwrite += write[i];

srread += read[i];

}

srwrite /= count;

srread /= count;

wrdis = abs(pow(wrm, 2) - wrm2);

rdis = abs(pow(rm, 2) - rm2);

printToFile(type, sizeType, count, size, rdis, read, write, srread, srwrite, wrdis, 1);

}

}

else {

bool flash = true;

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

if (num > 1) {

if (j == 1) {

cout << "Запускать тест flash? 1 - да, 2 - нет" << endl;

cin >> userInput;

if (userInput == 2) {

flash = false;

}

}

type = memoryType[j];

size = 4194304;

}

if ((type == "flash" || type == "SSD") && flash){

cout << "Введите адрес носителя:";

cin >> address;

}

for (int k = 0; k < numStep && flash; k++) {

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

testMemory(address, size, write[i], read[i]);

wrm += write[i] / count;

wrm2 += pow(write[i], 2) / count;

rm += read[i] / count;

rm2 += pow(read[i], 2) / count;

srwrite += write[i];

srread += read[i];

}

srwrite /= count;

srread /= count;

wrdis = abs(pow(wrm, 2) - wrm2);

rdis = abs(pow(rm, 2) - rm2);

printToFile(type, sizeType, count, size, rdis, read, write, srread, srwrite, wrdis, 1);

if (numStep == 20) {

size += 4194304;

wrm2 = 0, wrm = 0, wrdis, rm = 0, rm2 = 0, rdis = 0, srwrite = 0, srread = 0;

}

}

}

}

}