ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ВТ

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Оценка характеристик персонального компьютера» по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. АММ2-24

Ириков Евгений Алексеевич

Проверил: к.т.н., доцент Кафедры ВТ Перышкова Евгения Николаевна

Новосибирск 2024

Содержание

Постановка задачи

Разработать программу (benchmark) для оценки производительности подсистемы памяти.

1. Написать программум(функцию) на языке C/C++/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) 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;Write Time;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 [Мб/сек.]*

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

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

2. Написать программу(функцию) на языке C/C++/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.

** Для HDD|SSD и flash оценить влияние размера буфера (BufferSize) на пропускную способность памяти.*

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;

*4) * Зависимость пропускной способности памяти от размера буфера для HDD|SSD и flash памяти;*

Выполнение работы

Перед оценкой производительности подсистем памяти нужно определить размера кеш-линиии. Для нашей системы это:

L1 - 386 Kb
L2 - 1 Mb
L3 6 Mb

Запуск программы

run.sh 1 course\Computing systems\3 lab\run.sh

```
#!/bin/bash

echo "MemoryType ; BlockSize ; ElementType ; BufferSize ;\
 LaunchNum ; Timer; WriteTime; AverageWriteTime;\
 WriteBandwidth; AbsError(write); RelError(write);\
 ReadTime; AverageReadTime; ReadBandwidth; AbsError(read);\
 RelError(read)" > result.csv

g++ bm.cpp -o benchmark

./benchmark -m RAM -l 5 -b 386Kb
./benchmark -m RAM -l 5 -b 1Mb
./benchmark -m RAM -l 5 -b 6Mb
./benchmark -m RAM -l 5 -b 7Mb

for ((i=1; i <= 20; i++))
do
    let m=$i*4*1024*1024
    ./benchmark -m SSD -l 5 -b $m
done

for ((i=1; i <= 20; i++))
do
    let m=$i*4*1024*1024
    ./benchmark -m flash -l 5 -b $m
done
```

Bash-скрипт

Где параметр после названия исполняемого файла — это вид тестируемой памяти, допустимые значения RAM, HDD, SSD, flash.

Для обозначения размера блока памяти доступны наименования K – килобайты, M – мегабайты, ничего - байты

После блока памяти можно указать количество испытаний.

Результат работы

Запуск программы

```
PS C:\labs\MAGISTER\1 course\Computing systems\3 lab> bash run.sh
```

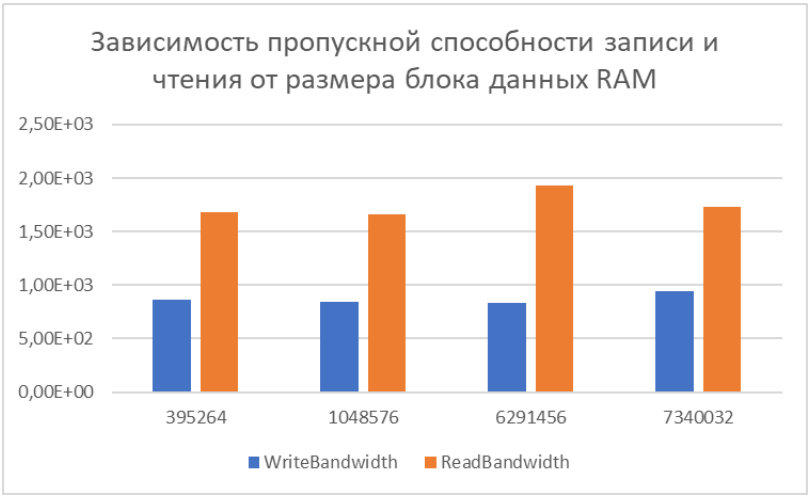
Для первого и второго задания

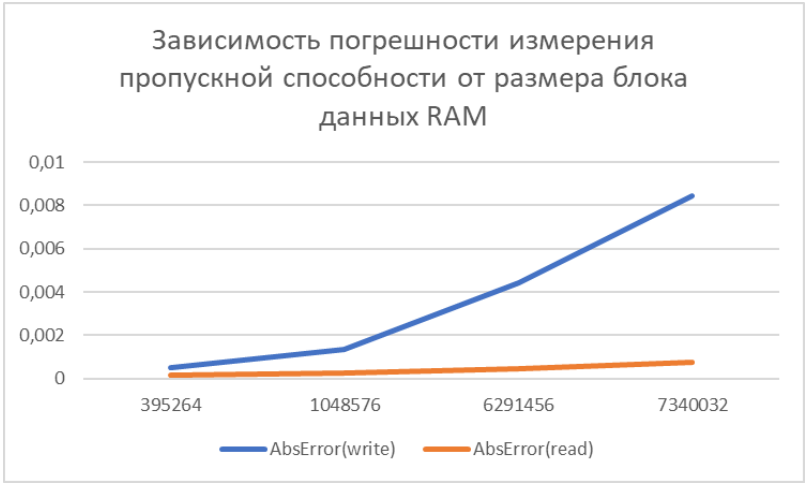
	I	J	K	L	M	N	O	P
1	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
2	8,65E+02	0,0005212	100,038	0,000205	0,00023488	1,68E+03	1,38E-04	58,7193
3	8,42E+02	0,00136212	105,006	0,000664	0,00063104	1,66E+03	2,74E-04	43,3507
4	8,31E+02	0,0044107	64,4962	0,003099	0,00325264	1,93E+03	0,00048434	14,8907
5	9,42E+02	0,00843816	93,4516	0,0037	0,0042433	1,73E+03	0,0007391	17,418
6	1,08E+02	0,00984936	22,3214	0,022285	0,0235905	1,78E+02	0,0024761	10,4962
7	1,05E+02	0,00754024	9,62311	0,049542	0,0512094	1,64E+02	0,0016676	3,25643
8	7,36E+01	0,00998892	8,28165	0,08741	0,0814101	1,55E+02	0,0139878	17,1819
9	7,83E+01	0,0264548	14,9921	0,121456	0,114691	1,46E+02	0,0223543	19,4909
10	7,62E+01	0,0631803	26,4746	0,187173	0,129709	1,62E+02	0,0574638	44,3021
11	7,28E+01	0,0149793	6,16028	0,132581	0,158285	1,59E+02	0,0793241	50,1149
12	1,05E+02	0,0591897	18,7929	0,162143	0,177889	1,65E+02	0,0569493	32,0139
13	9,24E+01	0,0263784	7,97929	0,40668	0,326363	1,03E+02	0,0803173	24,6098
14	7,95E+01	0,0621372	15,2838	0,250311	0,266341	1,42E+02	0,0161977	6,08156
15	9,01E+01	0,196272	31,1392	0,259368	0,267842	1,57E+02	0,00847368	3,16369
16	9,68E+01	0,0445055	10,0485	0,285818	0,285169	1,62E+02	0,00337616	1,18392
17	9,81E+01	0,0123128	2,77817	0,309789	0,315393	1,60E+02	0,00718704	2,27876
18	9,89E+01	0,0464752	9,04742	0,333396	0,338208	1,61E+02	0,00628942	1,85963
19	9,89E+01	0,0441002	7,47586	0,366226	0,368825	1,59E+02	0,0141808	3,84486
20	8,18E+01	0,037922	6,19063	0,428363	0,415035	1,52E+02	0,0133277	3,21123
21	9,21E+01	0,0845116	12,8827	0,45162	0,478074	1,40E+02	0,0528704	11,059
22	7,62E+01	0,115533	16,0246	0,477574	0,472806	1,51E+02	0,0122805	2,59736
23	9,25E+01	0,305467	38,6511	0,504844	0,487221	1,55E+02	0,0213868	4,38954
24	9,05E+01	0,563886	58,3502	0,504376	0,507679	1,57E+02	0,0119818	2,36012
25	9,42E+01	0,268058	31,0251	0,5151	0,523994	1,60E+02	0,0203256	3,87897
26	1,85E+02	0,00222752	10,7191	0,008254	0,00814134	5,15E+02	0,00011286	1,38626
27	1,87E+02	0,00263416	6,46008	0,016289	0,0162963	5,15E+02	0,00011918	0,731333
28	1,89E+02	0,00305032	5,05168	0,024157	0,0246004	5,11E+02	0,000443	1,80078
29	1,87E+02	0,0023654	2,89211	0,036871	0,0382562	4,39E+02	0,0160521	41,9595
30	1,85E+02	0,00633912	5,96315	0,053992	0,048719	4,30E+02	0,013085	26,8582
31	1,82E+02	0,00770402	6,42647	0,049486	0,0541079	4,65E+02	0,00961298	17,7663
32	1,90E+02	0,00891412	6,10943	0,058798	0,0590187	4,97E+02	0,00294438	4,98889
33	1,86E+02	0,0130751	7,78083	0,0676	0,0670063	5,01E+02	0,00122844	1,83332
34	1,84E+02	0,00864228	4,69853	0,074204	0,074958	5,04E+02	0,00133522	1,78129
35	1,89E+02	0,0164356	7,93706	0,08226	0,0819718	5,12E+02	0,00103268	1,2598
36	1,87E+02	0,0132367	5,68826	0,095636	0,094045	4,91E+02	0,00159046	1,69117
37	1,83E+02	0,0121893	4,89489	0,100292	0,103566	4,86E+02	0,0032735	3,1608
38	1,84E+02	0,0125587	4,77032	0,107646	0,109849	4,96E+02	0,00426676	3,88421
39	1,82E+02	0,0068404	2,37173	0,11963	0,125115	4,69E+02	0,00696036	5,56319
40	1,77E+02	0,0217461	7,00027	0,123621	0,136548	4,61E+02	0,012927	9,46697
41	1,67E+02	0,0470207	13,6331	0,135935	0,142091	4,72E+02	0,0242843	17,0906
42	1,71E+02	0,0114123	3,25431	0,173226	0,162947	4,38E+02	0,0199843	12,2644
43	1,59E+02	0,0224404	5,7496	0,151077	0,154832	4,88E+02	0,0119553	7,72143
44	1,78E+02	0,0196923	5,01093	0,162477	0,162696	4,90E+02	0,007794	4,79051
45	1,83E+02	0,247742	46,1976	0,187281	0,173694	4,83E+02	0,0135864	7,82205

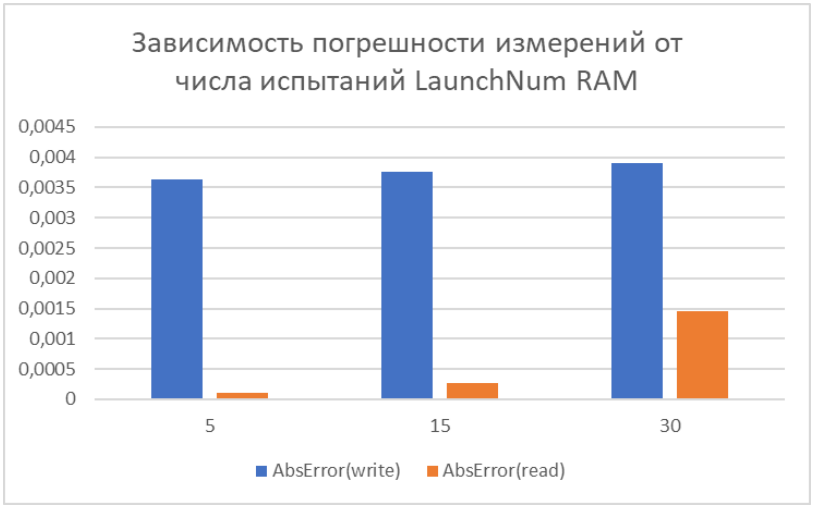
Для третьего задания

MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
RAM	3145728	int	3145728	5	clock_gettime()	0,0024313	0,0049484	635,706	0,0036323	73,4035	0,001772	0,00184652	1703,6	0,00010898	5,90191
RAM	3145728	int	3145728	15	clock_gettime()	0,0023367	0,0030139	1043,74	0,0037624	124,835	0,002051	0,00193061	1629,39	0,000275487	14,2694
RAM	3145728	int	3145728	30	clock_gettime()	0,0023376	0,0027879	1128,35	0,0039072	140,148	0,001943	0,00201387	1562,03	0,00145233	72,1164
SSD	41943040	int	41943040	5	clock_gettime()	0,420718	0,42253	99,2665	0,00274626	0,649957	0,256813	0,256364	163,608	0,00076766	0,299442
SSD	41943040	int	41943040	15	clock_gettime()	0,424833	0,426499	98,3426	0,00869539	2,03878	0,26181	0,264387	158,643	0,0171883	6,50117
SSD	41943040	int	41943040	30	clock_gettime()	0,420568	0,423847	98,9581	0,0108807	2,56713	0,259151	0,259077	161,894	0,0187352	7,23152
flash	41943040	int	41943040	5	clock_gettime()	0,229666	0,224081	187,178	0,00558528	2,49252	0,086654	0,0849986	493,455	0,00216794	2,55056
flash	41943040	int	41943040	15	clock_gettime()	0,222277	0,221669	189,215	0,00832862	3,75724	0,081946	0,0820822	510,988	0,00192038	2,33958
flash	41943040	int	41943040	30	clock_gettime()	0,244828	0,234702	178,707	0,0354507	15,1045	0,098268	0,0923106	454,369	0,0226038	24,4867

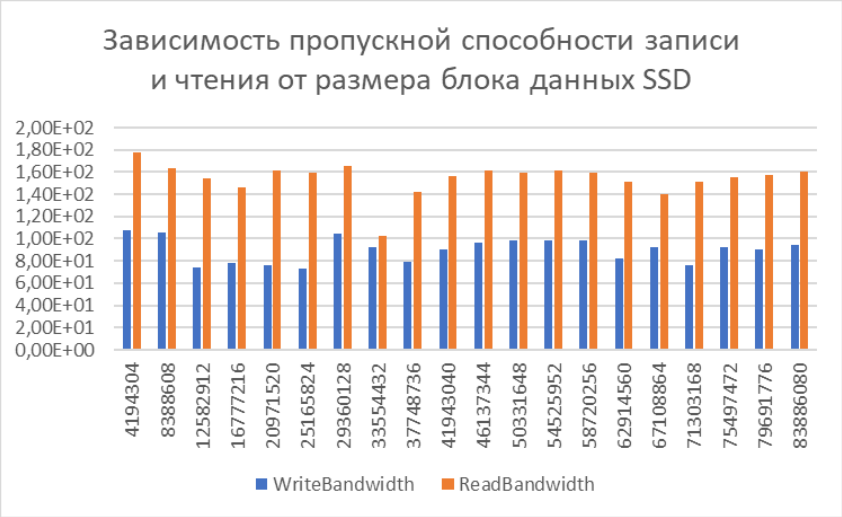
RAM

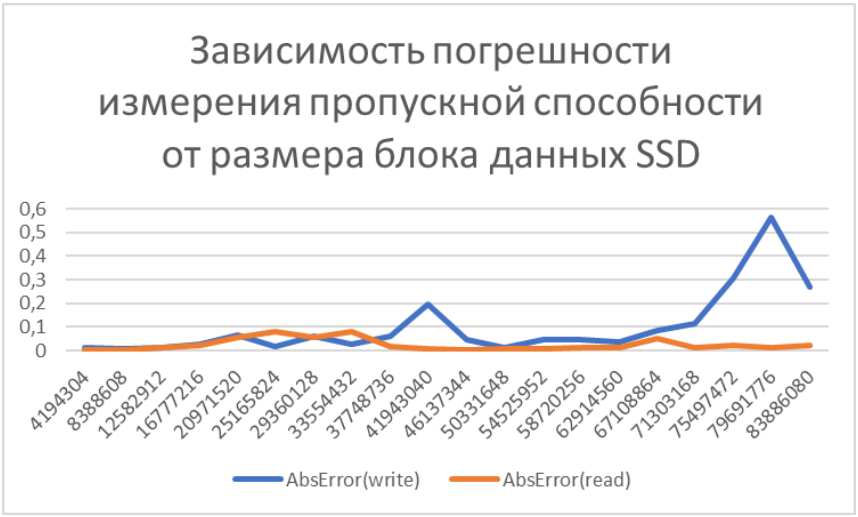

Зависимость пропускной способности записи и чтения от размера блока данных RAM
2,50E+03
2,00E+03
1,50E+03
1,00E+03
5,00E+02
0,00E+00
395264
1048576
6291456
7340032
WriteBandwidth
ReadBandwidth

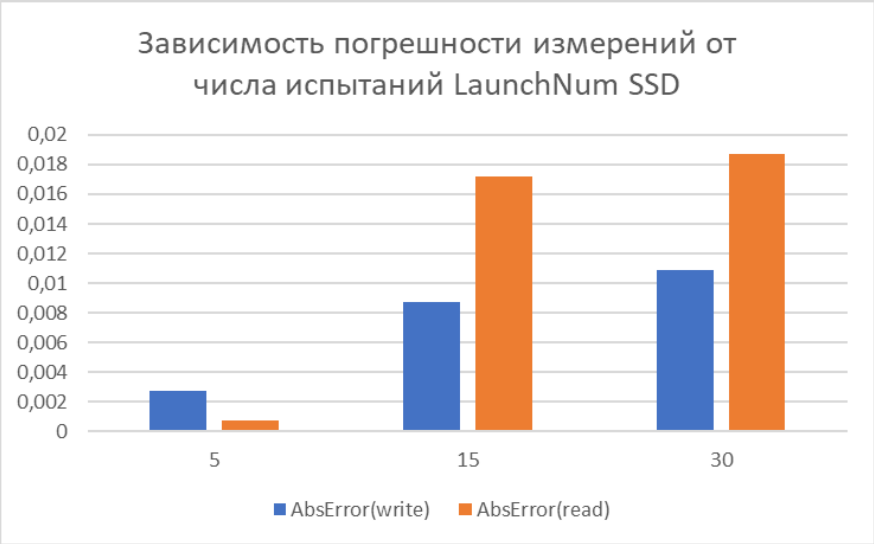

Зависимость погрешности измерения пропускной способности от размера блока данных RAM
0,01
0,008
0,006
0,004
0,002
0
395264
1048576
6291456
7340032
AbsError(write)
AbsError(read)


Зависимость погрешности измерений от числа испытаний LaunchNum RAM
0,0045
0,004
0,0035
0,003
0,0025
0,002
0,0015
0,001
0,0005
0
5
15
30
AbsError(write)
AbsError(read)

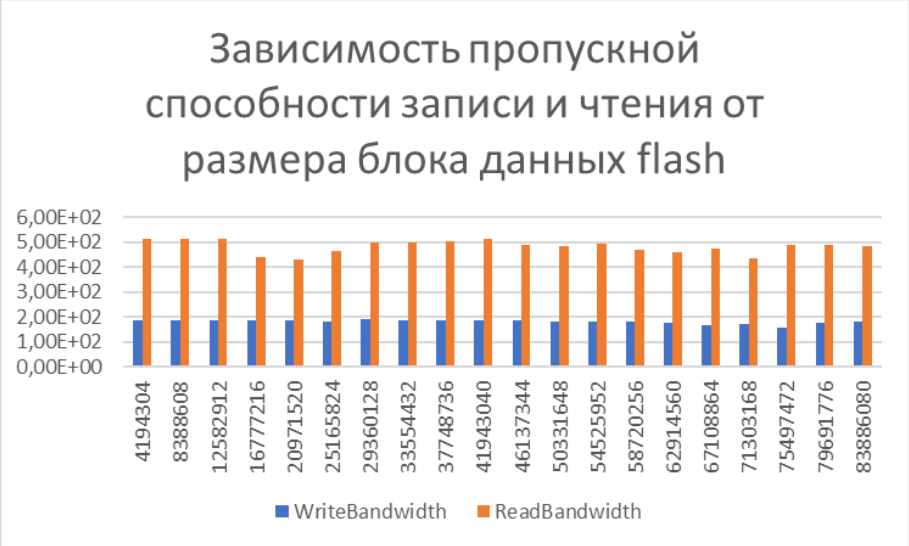
SSD

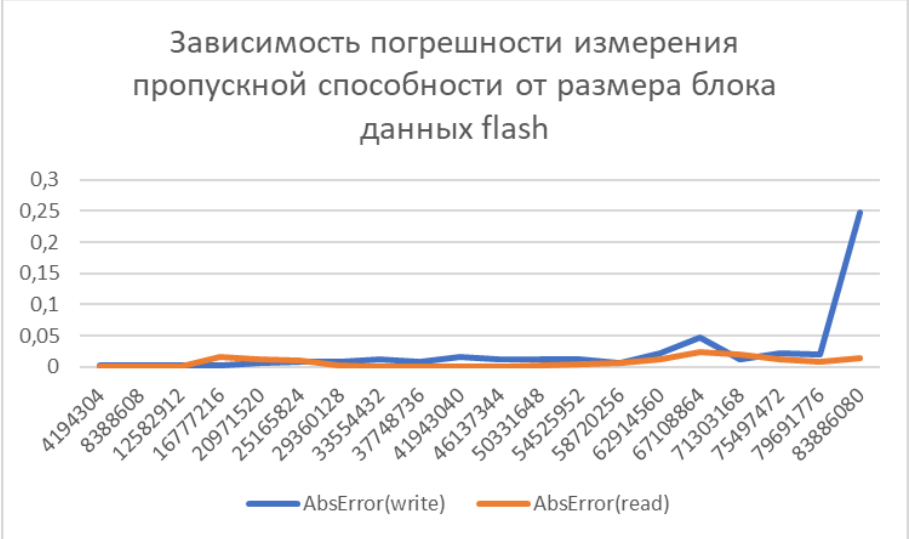

Зависимость пропускной способности записи и чтения от размера блока данных SSD
2,00E+02
1,80E+02
1,60E+02
1,40E+02
1,20E+02
1,00E+02
8,00E+01
6,00E+01
4,00E+01
2,00E+01
0,00E+00
4194304
8388608
12582912
16777216
20971520
25165824
29360128
33554432
37748736
41943040
46137344
50331648
54525952
58720256
62914560
67108864
71303168
75497472
79691776
83886080
WriteBandwidth
ReadBandwidth

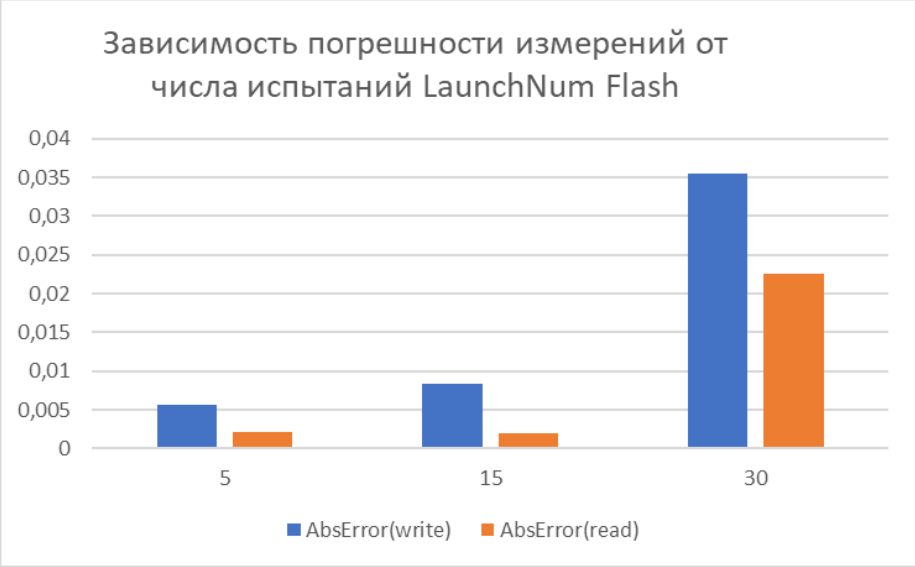

Зависимость погрешности измерения пропускной способности от размера блока данных SSD
0,6
0,5
0,4
0,3
0,2
0,1
0
4194304
8388608
12582912
16777216
20971520
25165824
29360128
33554432
37748736
41943040
46137344
50331648
54525952
58720256
62914560
67108864
71303168
75497472
79691776
83886080
AbsError(write)
AbsError(read)


Зависимость погрешности измерений от числа испытаний LaunchNum SSD
0,02
0,018
0,016
0,014
0,012
0,01
0,008
0,006
0,004
0,002
0
5
15
30
AbsError(write)
AbsError(read)

FLASH


Зависимость пропускной способности записи и чтения от размера блока данных flash
6,00E+02
5,00E+02
4,00E+02
3,00E+02
2,00E+02
1,00E+02
0,00E+00
4194304
8388608
12582912
16777216
20971520
25165824
29360128
33554432
37748736
41943040
46137344
50331648
54525952
58720256
62914560
67108864
71303168
75497472
79691776
83886080
WriteBandwidth
ReadBandwidth


Зависимость погрешности измерения пропускной способности от размера блока данных flash
0,3
0,25
0,2
0,15
0,1
0,05
0
4194304
8388608
12582912
16777216
20971520
25165824
29360128
33554432
37748736
41943040
46137344
50331648
54525952
58720256
62914560
67108864
71303168
75497472
79691776
83886080
AbsError(write)
AbsError(read)


Зависимость погрешности измерений от числа испытаний LaunchNum Flash
0,04
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
5
15
30
AbsError(write)
AbsError(read)

Приложение

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <ctime>
#include <cmath>
#include <Windows.h>

// L1 386 Kb
// L2 1 Mb
// L3 6 Mb

//clock_gettime in windows//
LARGE_INTEGER getFILETIMEoffset()
{
    SYSTEMTIME s;
    FILETIME f;
    LARGE_INTEGER t;

    s.wYear = 1970;
    s.wMonth = 1;
    s.wDay = 1;
    s.wHour = 0;
    s.wMinute = 0;
    s.wSecond = 0;
    s.wMilliseconds = 0;
    SystemTimeToFileTime(&s, &f);
    t.QuadPart = f.dwHighDateTime;
    t.QuadPart <<= 32;
    t.QuadPart |= f.dwLowDateTime;
    return (t);
}

int clock_gettime(int X, struct timeval *tv)
{
    LARGE_INTEGER           t;
    FILETIME            f;
    double                  microseconds;
    static LARGE_INTEGER    offset;
    static double           frequencyToMicroseconds;
    static int              initialized = 0;
    static BOOL             usePerformanceCounter = 0;

    if (!initialized) {
        LARGE_INTEGER performanceFrequency;
        initialized = 1;
        usePerformanceCounter = QueryPerformanceFrequency(&performanceFrequency);
        if (usePerformanceCounter) {
            QueryPerformanceCounter(&offset);
            frequencyToMicroseconds = (double)performanceFrequency.QuadPart / 1000000.;
        } else {
            offset = getFILETIMEoffset();
            frequencyToMicroseconds = 10.;
        }
    }
    if (usePerformanceCounter) QueryPerformanceCounter(&t);
    else {
        GetSystemTimeAsFileTime(&f);
        t.QuadPart = f.dwHighDateTime;
        t.QuadPart <<= 32;
        t.QuadPart |= f.dwLowDateTime;
    }

    t.QuadPart -= offset.QuadPart;
    microseconds = (double)t.QuadPart / frequencyToMicroseconds;
    t.QuadPart = microseconds;
    tv->tv_sec = t.QuadPart / 1000000;
    tv->tv_usec = t.QuadPart % 1000000;
    return (0);
}
//clock_gettime in windows//
```

```
typedef unsigned int uint;

#define NANOS_IN_SEC 1000000000

double* test_ram_w(uint block_size, uint launch_count)
{
    srand(time(0));
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
        uint arr_size = block_size / sizeof(uint);
        uint* initial_arr = new uint [arr_size];
        uint* test_arr = new uint[arr_size];

        for (uint i = 0; i < arr_size; i++) initial_arr[i] = rand() % 100;

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++) test_arr[i] = initial_arr[i];
        clock_gettime (0, &end);

        //std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        //std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;
        std::cout << result[launch] << '\n';
        delete test_arr;
    }

    return result;
}

double* test_ssd_w(uint block_size, uint launch_count)
{
    srand(time(0));
    uint arr_size = block_size / sizeof(uint);
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
        double duration = 0;

        std::ofstream out("mem_test", std::ios::binary | std::ios::out);

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++)
        {
            uint randval = rand() % 100;
            out.write((char*) &randval, sizeof(uint));
        }
        clock_gettime (0, &end);

        out.close();

        if (end.tv_sec - begin.tv_sec > 0)
            duration = (double)((end.tv_sec - begin.tv_sec) * NANOS_IN_SEC + end.tv_usec -
begin.tv_usec) / NANOS_IN_SEC;
        else
            duration = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;

        std::cout << "duration: " << duration << '\n';
        // std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        // std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = duration;
    }

    return result;
}

double* test_flash_w(uint block_size, uint launch_count)
{
```

```
    srand(time(0));
    uint arr_size = block_size / sizeof(uint);
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
        double duration = 0;

        std::ofstream out("/media/zer0chance/Transcend/mem_test", std::ios::binary | std::ios::out);

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++)
        {
            uint randval = rand() % 100;
            out.write((char*) &randval, sizeof(uint));
        }
        clock_gettime (0, &end);

        out.close();

        if (end.tv_sec - begin.tv_sec > 0)
            duration = (double)((end.tv_sec - begin.tv_sec) * NANOS_IN_SEC + end.tv_usec -
begin.tv_usec) / NANOS_IN_SEC;
        else
            duration = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;

        std::cout << "duration: " << duration << '\n';
        // std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        // std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = duration;
    }

    return result;
}

double* test_ram_r(uint block_size, uint launch_count)
{
    srand(time(0));
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
        uint arr_size = block_size / sizeof(uint);
        uint* test_arr = new uint [arr_size];
        uint* initial_arr = new uint[arr_size];

        for (uint i = 0; i < arr_size; i++) initial_arr[i] = rand() % 100;

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++) test_arr[i] = initial_arr[i];
        clock_gettime (0, &end);

        //std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        //std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;
        std::cout << result[launch] << '\n';
        delete initial_arr;
    }

    return result;
}

double* test_ssd_r(uint block_size, uint launch_count)
{
    srand(time(0));
    uint arr_size = block_size / sizeof(uint);
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
```

```
        double duration = 0;

        std::ifstream ifs("mem_test", std::ios::binary | std::ios::in);

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++)
        {
            uint val;
            ifs.read((char*) &val, sizeof(uint));
        }
        clock_gettime (0, &end);

        ifs.close();

        if (end.tv_sec - begin.tv_sec > 0)
            duration = (double)((end.tv_sec - begin.tv_sec) * NANOS_IN_SEC + end.tv_usec -
begin.tv_usec) / NANOS_IN_SEC;
        else
            duration = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;

        std::cout << "duration: " << duration << '\n';
        // std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        // std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = duration;
    }

    return result;
}

double* test_flash_r(uint block_size, uint launch_count)
{
    srand(time(0));
    uint arr_size = block_size / sizeof(uint);
    double* result = new double[launch_count];
    timeval begin, end;

    for (int launch = 0; launch < launch_count; launch++)
    {
        double duration = 0;

        std::ifstream ifs("/media/zer0chance/Transcend/mem_test", std::ios::binary | std::ios::in);

        clock_gettime (0, &begin);
        for (uint i = 0; i < arr_size; i++)
        {
            uint val;
            ifs.read((char*) &val, sizeof(uint));
        }
        clock_gettime (0, &end);

        ifs.close();

        if (end.tv_sec - begin.tv_sec > 0)
            duration = (double)((end.tv_sec - begin.tv_sec) * NANOS_IN_SEC + end.tv_usec -
begin.tv_usec) / NANOS_IN_SEC;
        else
            duration = (double)(end.tv_usec - begin.tv_usec) / NANOS_IN_SEC;

        std::cout << "duration: " << duration << '\n';
        // std::cout << "sec: " << end.tv_sec - begin.tv_sec << '\n';
        // std::cout << "nanosec: " << end.tv_usec - begin.tv_usec << '\n';
        result[launch] = duration;
    }

    return result;
}

inline double count_avgTime(double* result, uint launch_count)
{
    double sum = 0;

    for (int i = 0; i < launch_count; i++)
        sum += result[i];
```

```
    return sum / launch_count;
}

inline double count_absErr(double* result, uint launch_count, double avgWrtTime)
{
    double max = fabs(result[0] - avgWrtTime);

    for (int i = 1; i < launch_count; i++)
        if (fabs(result[i] - avgWrtTime) > max)
            max = fabs(result[i] - avgWrtTime);

    return max;
}

int main(int argc, char** argv)
{
    std::string mem_type("");
    uint block_size(0);
    uint launch_count(0);

    for (int i = 1; i < argc; i++)
    {
        if (!strcmp(argv[i], "-m") || !strcmp(argv[i], "--memory-type")) {
            if(i + 1 < argc) {
                mem_type = argv[i + 1];
                i++;
            }
            else {
                std::cout << "Memory type specified incorrectly\n";
                return 1;
            }
        } else if (!strcmp(argv[i], "-b") || !strcmp(argv[i], "--block-size")) {
            if(i + 1 < argc) {
                if (argv[i + 1][strlen(argv[i + 1]) - 1] != 'b') {   // not Mb or Kb -> Byte
                    block_size = strtol(argv[i + 1], NULL, 10);
                }
                else if (argv[i + 1][strlen(argv[i + 1]) - 2] == 'K') {  // Kb
                    block_size = strtol(argv[i + 1], NULL, 10) * 1024;
                }
                else if (argv[i + 1][strlen(argv[i + 1]) - 2] == 'M') {  // Mb
                    block_size = strtol(argv[i + 1], NULL, 10) * 1024 * 1024;
                } else {
                    std::cout << "Block size specified incorrectly\n";
                    return 1;
                }
                i++;
            }
            else {
                std::cout << "Block size specified incorrectly\n";
                return 1;
            }
        } else if (!strcmp(argv[i], "-l") || !strcmp(argv[i], "--launch-count")) {
            if(i + 1 < argc) {
                launch_count = strtol(argv[i + 1], NULL, 10);
                i++;
            } else {
                std::cout << "Launch count specified incorrectly\n";
                return 1;
            }
        } else {
            std::cout << "Unknown option: " << argv[i] << '\n';
            return 1;
        }
    }

    if (launch_count == 0) {
        std::cout << "Launch count is not specified\n";
        return 1;
    }
    if (block_size == 0) {
        std::cout << "Block size is not specified\n";
        return 1;
```

```
    }
    if (mem_type == "") {
        std::cout << "Memory type is not specified\n";
        return 1;
    }

    double* result_w = nullptr;
    double* result_r = nullptr;

    if (mem_type == "RAM") {
        result_w = test_ram_w(block_size, launch_count);
        result_r = test_ram_r(block_size, launch_count);
    } else if (mem_type == "HDD" || mem_type == "SSD") {
        result_w = test_ssd_w(block_size, launch_count);
        result_r = test_ssd_r(block_size, launch_count);
    } else if (mem_type == "flash") {
        result_w = test_flash_w(block_size, launch_count);
        result_r = test_flash_r(block_size, launch_count);
    } else {
        std::cout << "Unknown memory type\n";
        return 1;
    }

    std::ofstream out("result.csv", std::ios::app);

    double avgWrtTime = count_avgTime(result_w, launch_count);
    double absErr_w = count_absErr(result_w, launch_count, avgWrtTime);
    double realErr_w = (absErr_w / avgWrtTime) * 100;

    out << mem_type << ';' << block_size << ";int;" << block_size << ';'
        << launch_count << ";clock_gettime();" << result_w[launch_count - 1]
        << ';' << avgWrtTime << ';' << (block_size / avgWrtTime) / 1000000
        << ';' << absErr_w << ';' << realErr_w << ';';

    double avgReadTime = count_avgTime(result_r, launch_count);
    double absErr_r = count_absErr(result_r, launch_count, avgReadTime);
    double realErr_r = (absErr_r / avgReadTime) * 100;

    out << result_r[launch_count - 1] << ';' << avgReadTime << ';'
        << (block_size / avgReadTime) / 1000000 << ';' << absErr_r << ';'
        << realErr_r << '\n';

    delete result_w;
    delete result_r;
    out.close();
}
```