Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации Сибирский Государственный Университет

Телекоммуникаций и Информатики СибГУТИ

Кафедра Вычислительных систем

Лабораторная работа №3

По дисциплине “Архитектура вычислительных систем”

Выполнил:

Студент группы ИВ-921

Черемисин И.И.

Работу проверил:

Ассистент кафедры ВС

Петухова Я.В.

Новосибирск 2021

# **Задание**

Разработать программу (benchmark) для оценки производительности подсистемы памяти.

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) \* 106 [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) \* 106 [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. \* Для HDD|SSD и flash оценить влияние размера буфера (BufferSize) на пропускную способность памяти. 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;

# **Результат Работы**

­

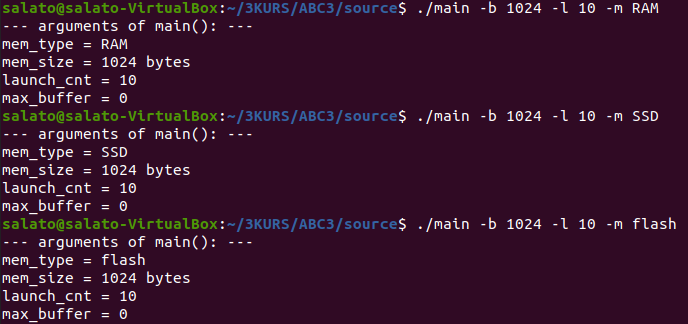


Рис.1 Результат работы программы

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MemoryType | BlockSize | ElementType | BufferSize | LaunchNum | Timer | WriteTime | AverageWriteTime | WriteBandwidth | AbsError(write) | RelError(write) | ReadTime | AverageReadTime | ReadBandwidth | AbsError(read) | RelError(read) |
| RAM | 64 | uint8\_t | 1 | 1 | clock() | 2E-06 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% | 2E-06 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% |
| RAM | 64 | uint8\_t | 1 | 2 | clock() | 1E-06 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% | 1E-06 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% |
| RAM | 64 | uint8\_t | 1 | 3 | clock() | 0 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% | 0 | 1E-06 | 64000000000000 | 8.164966E-07 | 6.666667e-05% |
| RAM | 32768 | uint8\_t | 1 | 1 | clock() | 5.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% | 5.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% |
| RAM | 32768 | uint8\_t | 1 | 2 | clock() | 4.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% | 4.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% |
| RAM | 32768 | uint8\_t | 1 | 3 | clock() | 4.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% | 4.4E-05 | 4.733333E-05 | 692281700000000 | 4.714045E-06 | 4.694836e-05% |
| RAM | 524288 | uint8\_t | 1 | 1 | clock() | 0.001172 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% | 0.001172 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% |
| RAM | 524288 | uint8\_t | 1 | 2 | clock() | 0.000957 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% | 0.000957 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% |
| RAM | 524288 | uint8\_t | 1 | 3 | clock() | 0.001011 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% | 0.001011 | 0.001046667 | 500912100000000 | 9.132482E-05 | 7.968365e-04% |
| RAM | 16777216 | uint8\_t | 1 | 1 | clock() | 0.031886 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% | 0.031886 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% |
| RAM | 16777216 | uint8\_t | 1 | 2 | clock() | 0.030926 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% | 0.030926 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% |
| RAM | 16777216 | uint8\_t | 1 | 3 | clock() | 0.030291 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% | 0.030291 | 0.03103433 | 540601800000000 | 0.0006556464 | 1.385150e-03% |
| RAM | 17301504 | uint8\_t | 1 | 1 | clock() | 0.030475 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% | 0.030475 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% |
| RAM | 17301504 | uint8\_t | 1 | 2 | clock() | 0.034582 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% | 0.034582 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% |
| RAM | 17301504 | uint8\_t | 1 | 3 | clock() | 0.031555 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% | 0.031555 | 0.032204 | 537247100000000 | 0.001738345 | 9.383437e-03% |
| SSD | 4194304 | uint8\_t | 4194304 | 1 | clock() | 0.003376 | 0.003376 | 1242389000000000 | 0 | 0.000000e+00% | 0.004013 | 0.004013 | 1.045179E+015 | 0 | 0.000000e+00% |
| SSD | 8388608 | uint8\_t | 8388608 | 1 | clock() | 0.00447 | 0.00447 | 1876646000000000 | 0 | 0.000000e+00% | 0.003231 | 0.003231 | 2596288000000000 | 0 | 0.000000e+00% |
| SSD | 12582912 | uint8\_t | 12582912 | 1 | clock() | 0.009614 | 0.009614 | 1308811000000000 | 0 | 0.000000e+00% | 0.006806 | 0.006806 | 1.848797E+015 | 0 | 0.000000e+00% |
| SSD | 16777216 | uint8\_t | 16777216 | 1 | clock() | 0.008398 | 0.008398 | 1997763000000000 | 0 | 0.000000e+00% | 0.004272 | 0.004272 | 3927251000000000 | 0 | 0.000000e+00% |
| SSD | 20971520 | uint8\_t | 20971520 | 1 | clock() | 0.013089 | 0.013089 | 1602225000000000 | 0 | 0.000000e+00% | 0.014744 | 0.014744 | 1422377000000000 | 0 | 0.000000e+00% |
| SSD | 25165824 | uint8\_t | 25165824 | 1 | clock() | 0.013777 | 0.013777 | 1826655000000000 | 0 | 0.000000e+00% | 0.021179 | 0.021179 | 1188244000000000 | 0 | 0.000000e+00% |
| SSD | 29360128 | uint8\_t | 29360128 | 1 | clock() | 0.016785 | 0.016785 | 1749188000000000 | 0 | 0.000000e+00% | 0.012919 | 0.012919 | 2272632000000000 | 0 | 0.000000e+00% |
| SSD | 33554432 | uint8\_t | 33554432 | 1 | clock() | 0.020653 | 0.020653 | 1624676000000000 | 0 | 0.000000e+00% | 0.01946 | 0.01946 | 1.724277E+015 | 0 | 0.000000e+00% |
| SSD | 37748736 | uint8\_t | 37748736 | 1 | clock() | 0.021043 | 0.021043 | 1793886000000000 | 0 | 0.000000e+00% | 0.016266 | 0.016266 | 2320714000000000 | 0 | 0.000000e+00% |
| SSD | 41943040 | uint8\_t | 41943040 | 1 | clock() | 0.025216 | 0.025216 | 1663350000000000 | 0 | 0.000000e+00% | 0.02149 | 0.02149 | 1951747000000000 | 0 | 0.000000e+00% |
| SSD | 46137344 | uint8\_t | 46137344 | 1 | clock() | 0.02499 | 0.02499 | 1846232000000000 | 0 | 0.000000e+00% | 0.024248 | 0.024248 | 1902728000000000 | 0 | 0.000000e+00% |
| SSD | 50331648 | uint8\_t | 50331648 | 1 | clock() | 0.028978 | 0.028978 | 1736892000000000 | 0 | 0.000000e+00% | 0.024326 | 0.024326 | 2.069047E+015 | 0 | 0.000000e+00% |
| SSD | 54525952 | uint8\_t | 54525952 | 1 | clock() | 0.032242 | 0.032242 | 1691147000000000 | 0 | 0.000000e+00% | 0.021284 | 0.021284 | 2561828000000000 | 0 | 0.000000e+00% |
| SSD | 58720256 | uint8\_t | 58720256 | 1 | clock() | 0.034066 | 0.034066 | 1723720000000000 | 0 | 0.000000e+00% | 0.02688 | 0.02688 | 2184533000000000 | 0 | 0.000000e+00% |
| SSD | 62914560 | uint8\_t | 62914560 | 1 | clock() | 0.036802 | 0.036802 | 1709542000000000 | 0 | 0.000000e+00% | 0.03271 | 0.03271 | 1923404000000000 | 0 | 0.000000e+00% |
| SSD | 67108864 | uint8\_t | 67108864 | 1 | clock() | 0.0474 | 0.0474 | 1415799000000000 | 0 | 0.000000e+00% | 0.035281 | 0.035281 | 1.902125E+015 | 0 | 0.000000e+00% |
| SSD | 71303168 | uint8\_t | 71303168 | 1 | clock() | 0.043783 | 0.043783 | 1628558000000000 | 0 | 0.000000e+00% | 0.03506 | 0.03506 | 2033747000000000 | 0 | 0.000000e+00% |
| SSD | 75497472 | uint8\_t | 75497472 | 1 | clock() | 0.046446 | 0.046446 | 1625489000000000 | 0 | 0.000000e+00% | 0.037187 | 0.037187 | 2030211000000000 | 0 | 0.000000e+00% |
| SSD | 79691776 | uint8\_t | 79691776 | 1 | clock() | 0.048308 | 0.048308 | 1649660000000000 | 0 | 0.000000e+00% | 0.038749 | 0.038749 | 2.056615E+015 | 0 | 0.000000e+00% |
| SSD | 83886080 | uint8\_t | 83886080 | 1 | clock() | 0.051395 | 0.051395 | 1632184000000000 | 0 | 0.000000e+00% | 0.040103 | 0.040103 | 2.091766E+015 | 0 | 0.000000e+00% |
| flash | 4194304 | uint8\_t | 4194304 | 1 | clock() | 0 | 0 | inf | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 8388608 | uint8\_t | 8388608 | 1 | clock() | 0 | 0 | inf | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 536870900000000 | 0 | 0.000000e+00% |
| flash | 12582912 | uint8\_t | 12582912 | 1 | clock() | 0 | 0 | inf | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 16777216 | uint8\_t | 16777216 | 1 | clock() | 0 | 0 | inf | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 20971520 | uint8\_t | 20971520 | 1 | clock() | 0.015625 | 0.015625 | 1342177000000000 | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 25165824 | uint8\_t | 25165824 | 1 | clock() | 0.015625 | 0.015625 | 1.610613E+015 | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 29360128 | uint8\_t | 29360128 | 1 | clock() | 0.03125 | 0.03125 | 939524100000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 1879048000000000 | 0 | 0.000000e+00% |
| flash | 33554432 | uint8\_t | 33554432 | 1 | clock() | 0.03125 | 0.03125 | 1073742000000000 | 0 | 0.000000e+00% | 0 | 0 | inf | 0 | 0.000000e+00% |
| flash | 37748736 | uint8\_t | 37748736 | 1 | clock() | 0.03125 | 0.03125 | 1207960000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 2415919000000000 | 0 | 0.000000e+00% |
| flash | 41943040 | uint8\_t | 41943040 | 1 | clock() | 0 | 0 | inf | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 2684355000000000 | 0 | 0.000000e+00% |
| flash | 46137344 | uint8\_t | 46137344 | 1 | clock() | 0.015625 | 0.015625 | 2952790000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 2952790000000000 | 0 | 0.000000e+00% |
| flash | 50331648 | uint8\_t | 50331648 | 1 | clock() | 0.015625 | 0.015625 | 3221225000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 3221225000000000 | 0 | 0.000000e+00% |
| flash | 54525952 | uint8\_t | 54525952 | 1 | clock() | 0.03125 | 0.03125 | 1744830000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 3489661000000000 | 0 | 0.000000e+00% |
| flash | 58720256 | uint8\_t | 58720256 | 1 | clock() | 0.0625 | 0.0625 | 939524100000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 3758096000000000 | 0 | 0.000000e+00% |
| flash | 62914560 | uint8\_t | 62914560 | 1 | clock() | 0.03125 | 0.03125 | 2.013266E+015 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 4.026532E+015 | 0 | 0.000000e+00% |
| flash | 67108864 | uint8\_t | 67108864 | 1 | clock() | 0.03125 | 0.03125 | 2147484000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 4294967000000000 | 0 | 0.000000e+00% |
| flash | 71303168 | uint8\_t | 71303168 | 1 | clock() | 0.046875 | 0.046875 | 1521134000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 4563403000000000 | 0 | 0.000000e+00% |
| flash | 75497472 | uint8\_t | 75497472 | 1 | clock() | 0.0625 | 0.0625 | 1207960000000000 | 0 | 0.000000e+00% | 0.03125 | 0.03125 | 2415919000000000 | 0 | 0.000000e+00% |
| flash | 79691776 | uint8\_t | 79691776 | 1 | clock() | 0.046875 | 0.046875 | 1700091000000000 | 0 | 0.000000e+00% | 0.03125 | 0.03125 | 2550137000000000 | 0 | 0.000000e+00% |
| Flash­­ | 83886080 | uint8\_t | 83886080 | 1 | clock() | 0.0625 | 0.0625 | 1342177000000000 | 0 | 0.000000e+00% | 0.015625 | 0.015625 | 5368709000000000 | 0 | 0.000000e+00% |

Таблица.1. Результаты испытаний программы

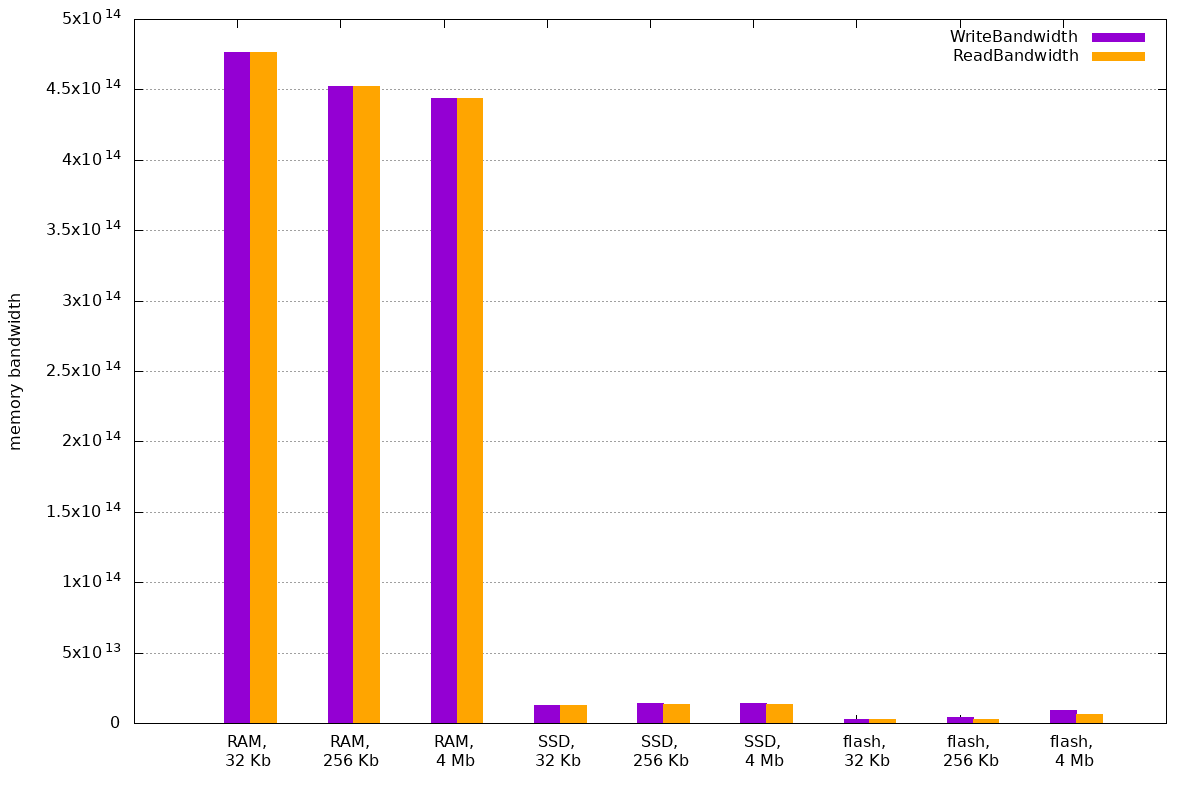


График 1. Зависимость пропускной способности записи и чтения от размера блока данных (benchmark) для оценки производительности подсистемы памяти.

График 2. Зависимость погрешности измерения пропускной способности от размера блока данных (benchmark) для разного типа памяти.

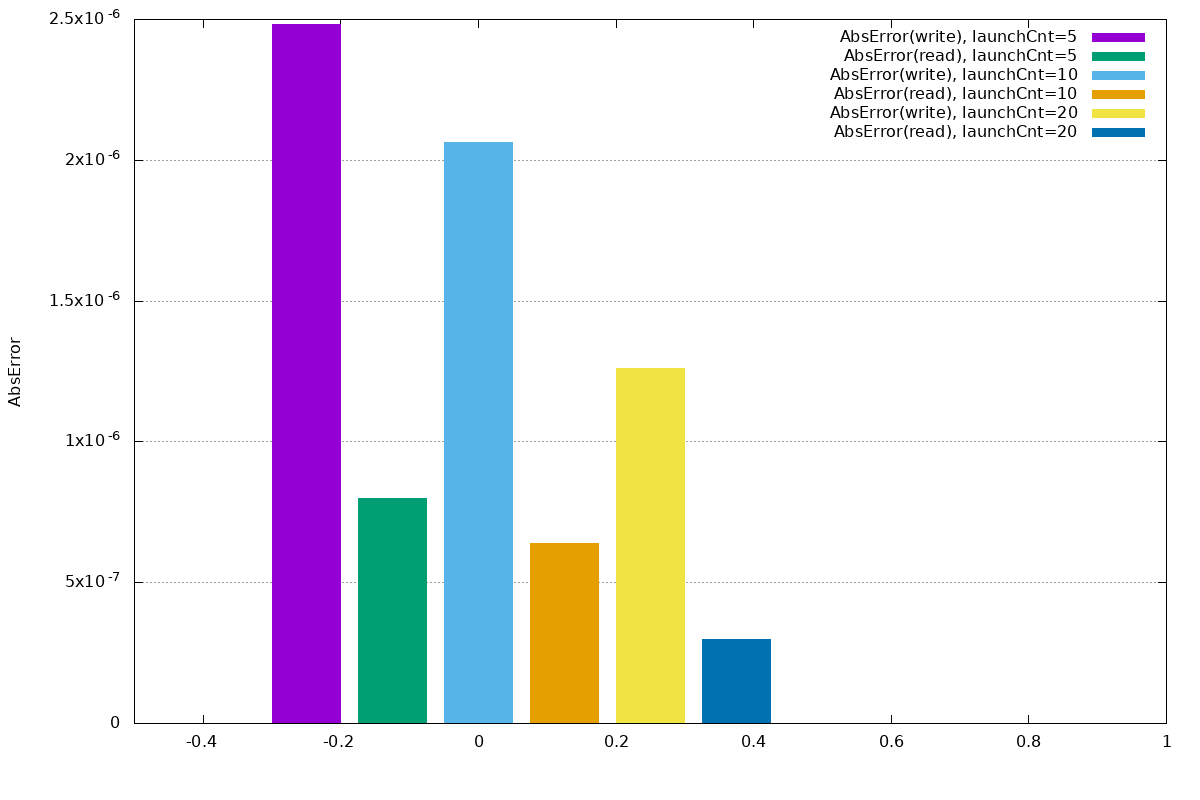
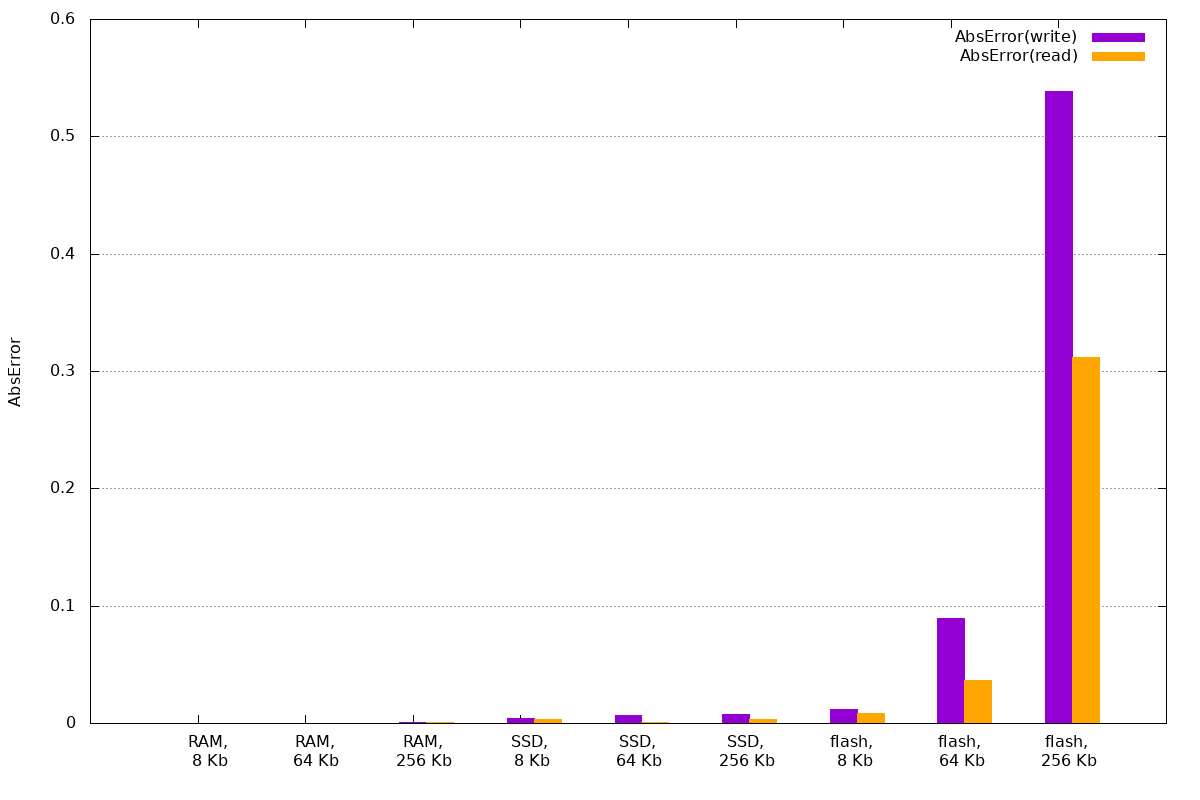


График 3. Зависимость погрешности измерений от числа испытаний LaunchNum.

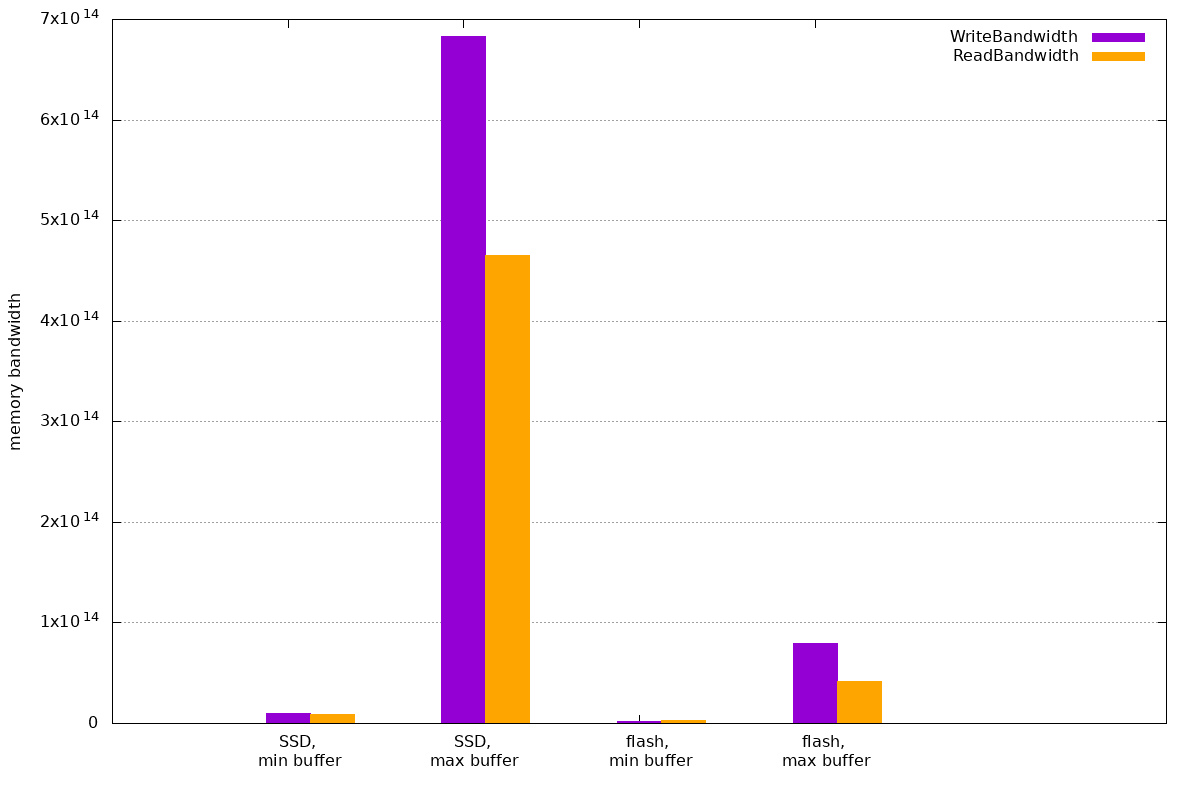


График 4. \* Зависимость пропускной способности памяти от размера буфера для SSD и flash памяти

# **Листинг**

**Main.cpp**

#include "memory.hpp"

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

**char**\* mem\_type = (**char**\*) malloc(**5**);

strcpy(mem\_type, "RAM**\0**");

**long** **long** mem\_size = **1024**;

**long** **long** launch\_cnt = **10**;

**bool** max\_buffer = false;

**if** (Process\_parameters(argc, argv, mem\_type, mem\_size,

launch\_cnt, max\_buffer) == **1**)

{

**return** **1**;

}

printf("--- arguments of main(): --- **\n**");

printf("mem\_type = %s **\n**", mem\_type);

printf("mem\_size = %lld bytes **\n**", mem\_size);

printf("launch\_cnt = %lld **\n**", launch\_cnt);

printf("max\_buffer = %d **\n**", max\_buffer);

srand(time(**0**));

Tests\_handler(mem\_type, launch\_cnt, mem\_size, max\_buffer);

free(mem\_type);

**return** **0**;

}

**Memory.cpp**

#include "memory.hpp"

**int** **Process\_parameters**(**int** argc, **char** \*argv[], **char**\* mem\_type,

**long** **long** &mem\_size, **long** **long** &launch\_cnt,

**bool** &max\_buffer)

{

**int** i;

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

{

**if** (strcmp("-m", argv[i]) == **0** ||

strcmp("--memory-type", argv[i]) == **0**)

{

i++;

**if** (strcmp("RAM", argv[i]) == **0** ||

strcmp("SSD", argv[i]) == **0** ||

strcmp("flash", argv[i]) == **0**)

{

strcpy(mem\_type, argv[i]);

}

**else** {

printf("Error in arguments of main(): incorrect value for --memory-type **\n**");

**return** **1**;

}

}

**else** **if** (strcmp("-b", argv[i]) == **0** ||

strcmp("--block-size", argv[i]) == **0**)

{

i++;

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

**int** coefficient = **1**;

**if** (argv[i][len-**1**] == 'b') {

**if** (argv[i][len-**2**] == 'K')

coefficient = **1024**;

**else** **if** (argv[i][len-**2**] == 'M')

coefficient = **1024** \* **1024**;

**else** {

printf("Error in arguments of main(): incorrect description of unit of measure byte **\n**");

**return** **1**;

}

strncpy(argv[i], argv[i], len - **2**);

}

mem\_size = atoll(argv[i]) \* coefficient;

}

**else** **if** (strcmp("-l", argv[i]) == **0** ||

strcmp("--launch-count", argv[i]) == **0**)

{

i++;

launch\_cnt = atoll(argv[i]);

**if** (launch\_cnt == **0**) {

printf("Error in arguments of main(): incorrect value for --launch-count **\n**");

**return** **1**;

}

}

**else** **if** (strcmp("--maximum-buffer", argv[i]) == **0**)

{

max\_buffer = true;

}

**else** {

printf("Error in arguments of main(): incorrect value (ind = %d) **\n**", i);

**return** **1**;

}

}

**return** **0**;

}

**int** **Test\_RAM**(**long** **long** mem\_size, **bool** max\_buffer,

**double** &time\_write, **double** &time\_read)

{

**uint8\_t**\* arr1 = (**uint8\_t**\*) malloc(mem\_size);

**long** **long** i;

**for** (i = **0**; i < mem\_size; i++) {

arr1[i] = rand() % **256**;

}

**uint8\_t**\* arr2 = (**uint8\_t**\*) malloc(mem\_size);

**clock\_t** start, stop;

**if** (max\_buffer) {

start = clock();

memcpy(arr2, arr1, mem\_size);

stop = clock();

}

**else** {

start = clock();

**for** (i = **0**; i < mem\_size; i++) {

arr2[i] = arr1[i];

}

stop = clock();

}

time\_write = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;

time\_read = time\_write;

free(arr1);

free(arr2);

**return** **0**;

}

**int** **Test\_storage\_device**(**char**\* mem\_type, **long** **long** mem\_size, **bool** max\_buffer,

**double** &time\_write, **double** &time\_read)

{

**uint8\_t**\* arr = (**uint8\_t**\*) malloc(mem\_size);

**long** **long** i;

**for** (i = **0**; i < mem\_size; i++) {

arr[i] = rand() % **256**;

}

**FILE** \*fp;

**if** (strcmp("SSD", mem\_type) == **0**) {

**if** ((fp = fopen("test\_SSD.txt", "w")) == NULL) {

printf("Error in Test\_storage\_device(): can't open file test\_SSD.txt **\n**");

**return** **1**;

}

}

**else** **if** (strcmp("flash", mem\_type) == **0**) {

**if** ((fp = fopen("/mnt/f/test\_flash.txt", "w")) == NULL) {

printf("Error in Test\_storage\_device(): can't open file test\_flash.txt in USB **\n**");

**return** **1**;

}

}

**clock\_t** start, stop;

**if** (max\_buffer) {

start = clock();

fwrite(arr, mem\_size, **1**, fp);

stop = clock();

}

**else** {

start = clock();

**for** (i = **0**; i < mem\_size; i++) {

fprintf(fp, "%hhu ", arr[i]);

}

stop = clock();

}

time\_write = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;

fclose(fp);

**if** (strcmp("SSD", mem\_type) == **0**) {

**if** ((fp = fopen("test\_SSD.txt", "r")) == NULL) {

printf("Error in Test\_storage\_device(): can't open file test\_SSD.txt **\n**");

**return** **1**;

}

}

**else** **if** (strcmp("flash", mem\_type) == **0**) {

**if** ((fp = fopen("/mnt/f/test\_flash.txt", "r")) == NULL) {

printf("Error in Test\_storage\_device(): can't open file test\_flash.txt in USB **\n**");

**return** **1**;

}

}

**if** (max\_buffer) {

start = clock();

fread(arr, mem\_size, **1**, fp);

stop = clock();

}

**else** {

start = clock();

**for** (i = **0**; i < mem\_size; i++) {

fscanf(fp, "%hhu", &arr[i]);

}

stop = clock();

}

time\_read = ((**double**)(stop - start)) / CLOCKS\_PER\_SEC;

free(arr);

fclose(fp);

**return** **0**;

}

**int** **Write\_to\_csv**(**char**\* mem\_type, **long** **long** launch\_cnt,

**long** **long** mem\_size, **bool** max\_buffer,

**double** time\_write[], **double** avg\_time\_write,

**double** abs\_error\_write, **double** rel\_error\_write,

**double** time\_read[], **double** avg\_time\_read,

**double** abs\_error\_read, **double** rel\_error\_read)

{

**FILE** \*fout;

**if** ((fout = fopen("../data/output.csv", "a")) == NULL) {

printf("Error in Write\_to\_csv(): can't open output.csv **\n**");

**return** **1**;

}

**for** (**long** **long** i = **0**; i < launch\_cnt; i++) {

fprintf(fout, "%s;", mem\_type);

fprintf(fout, "%lld;", mem\_size);

fprintf(fout, "uint8\_t;");

**if** (max\_buffer)

fprintf(fout, "%lld;", mem\_size);

**else**

fprintf(fout, "1;");

fprintf(fout, "%lld;", i+**1**);

fprintf(fout, "clock();");

fprintf(fout, "%e;", time\_write[i]);

fprintf(fout, "%e;", avg\_time\_write);

fprintf(fout, "%e;", (**double**)mem\_size / avg\_time\_write \* **1e6**);

fprintf(fout, "%e;", abs\_error\_write);

fprintf(fout, "%e%%;", rel\_error\_write);

fprintf(fout, "%e;", time\_read[i]);

fprintf(fout, "%e;", avg\_time\_read);

fprintf(fout, "%e;", (**double**)mem\_size / avg\_time\_read \* **1e6**);

fprintf(fout, "%e;", abs\_error\_read);

fprintf(fout, "%e%%;", rel\_error\_read);

fprintf(fout, "**\n**");

}

**return** **0**;

}

**int** **Tests\_handler**(**char**\* mem\_type, **long** **long** launch\_cnt,

**long** **long** mem\_size, **bool** max\_buffer)

{

**double** summand1\_write = **0**, summand2\_write = **0**;

**double** summand1\_read = **0**, summand2\_read = **0**;

**double** time\_write[launch\_cnt];

**double** time\_read[launch\_cnt];

**for** (**long** **long** i = **0**; i < launch\_cnt; i++) {

**if** (strncmp("RAM", mem\_type, **3**) == **0**) {

**if** (Test\_RAM(mem\_size, max\_buffer, time\_write[i], time\_read[i]) == **1**)

**return** **1**;

}

**else** {

**if** (Test\_storage\_device(mem\_type, mem\_size, max\_buffer,

time\_write[i], time\_read[i]) == **1**)

**return** **1**;

}

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

summand2\_write += time\_write[i];

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

summand2\_read += time\_read[i];

}

summand1\_write /= launch\_cnt;

summand2\_write /= launch\_cnt;

**double** avg\_time\_write = summand2\_write;

summand2\_write \*= summand2\_write;

**double** dispersion\_write = summand1\_write - summand2\_write;

**double** abs\_error\_write = sqrt(dispersion\_write);

**double** rel\_error\_write = dispersion\_write / avg\_time\_write \* **100**;

summand1\_read /= launch\_cnt;

summand2\_read /= launch\_cnt;

**double** avg\_time\_read = summand2\_read;

summand2\_read \*= summand2\_read;

**double** dispersion\_read = summand1\_read - summand2\_read;

**double** abs\_error\_read = sqrt(dispersion\_read);

**double** rel\_error\_read = dispersion\_read / avg\_time\_read \* **100**;

Write\_to\_csv(mem\_type, launch\_cnt, mem\_size, max\_buffer,

time\_write, avg\_time\_write,

abs\_error\_write, rel\_error\_write,

time\_read, avg\_time\_read,

abs\_error\_read, rel\_error\_read);

**return** **0**;

}

**Memory.hpp**

#ifndef MEMORY

#define MEMORY

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <stdint.h>

**int** **Process\_parameters**(**int** argc, **char** \*argv[], **char**\* mem\_type,

**long** **long** &mem\_size, **long** **long** &launch\_cnt,

**bool** &max\_buffer);

**int** **Tests\_handler**(**char**\* mem\_type, **long** **long** launch\_cnt,

**long** **long** mem\_size, **bool** max\_buffer);

#endif

**Makefile**

COMP = g++

FLAGS = -Wall -o

OBJS = main.cpp memory.cpp

.PHONY: clean run all

**all:** main

**main:**

$(COMP) $(OBJS) $(FLAGS) main -lm

**S21.sh**

#!/bin/bash

echo "MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;\

AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidth;AbsError(read);RelError(read);" > ../data/output.csv

cache\_line=$(getconf LEVEL1\_DCACHE\_LINESIZE)

cache\_lvl1=$(getconf LEVEL1\_DCACHE\_SIZE)

cache\_lvl2=$(getconf LEVEL2\_CACHE\_SIZE)

cache\_lvl3=$(getconf LEVEL3\_CACHE\_SIZE)

cache\_more=$(($cache\_lvl3 + $cache\_lvl2))

cd ../source

make

./../source/main --launch-count **3** --memory-type RAM --block-size $cache\_line

./../source/main --launch-count **3** --memory-type RAM --block-size $cache\_lvl1

./../source/main --launch-count **3** --memory-type RAM --block-size $cache\_lvl2

./../source/main --launch-count **3** --memory-type RAM --block-size $cache\_lvl3

./../source/main --launch-count **3** --memory-type RAM --block-size $cache\_more

**S22.sh**

#!/bin/bash

echo "MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;\

AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidth;AbsError(read);RelError(read);" > ../data/output.csv

mem\_size=**4**

cd ../source

make

**for** (( i=**1**; i <= **20**; i++ ))

**do**

./../source/main -l **1** -m SSD -b $mem\_size"Mb" --maximum-buffer

mem\_size=$(( $mem\_size + **4**))

done

**s31-32.sh**

#!/bin/bash

echo "MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;\

AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidth;AbsError(read);RelError(read);" > ../data/output.csv

cd ../source

make

./../source/main --launch-count **10** --memory-type RAM --block-size **8**Kb

./../source/main --launch-count **10** --memory-type RAM --block-size **64**Kb

./../source/main --launch-count **10** --memory-type RAM --block-size **256**Kb

./../source/main --launch-count **10** --memory-type SSD --block-size **8**Kb

./../source/main --launch-count **10** --memory-type SSD --block-size **64**Kb

./../source/main --launch-count **10** --memory-type SSD --block-size **256**Kb

./../source/main --launch-count **10** --memory-type flash --block-size **8**Kb

./../source/main --launch-count **10** --memory-type flash --block-size **64**Kb

./../source/main --launch-count **10** --memory-type flash --block-size **256**Kb

**S33.sh**

#!/bin/bash

echo "MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;\

AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidth;AbsError(read);RelError(read);" > ../data/output.csv

cd ../source

make

./../source/main --launch-count **5** --memory-type SSD --block-size **12**Kb --maximum-buffer

./../source/main --launch-count **10** --memory-type SSD --block-size **12**Kb --maximum-buffer

./../source/main --launch-count **20** --memory-type SSD --block-size **12**Kb --maximum-buffer

**Star.sh**

#!/bin/bash

echo "MemoryType;BlockSize;ElementType;BufferSize;LaunchNum;Timer;WriteTime;AverageWriteTime;WriteBandwidth;\

AbsError(write);RelError(write);ReadTime;AverageReadTime;ReadBandwidth;AbsError(read);RelError(read);" > ../data/output.csv

cd ../source

make

./../source/main --launch-count **10** --memory-type SSD --block-size **2**Kb

./../source/main --launch-count **10** --memory-type SSD --block-size **2**Kb --maximum-buffer

./../source/main --launch-count **10** --memory-type flash --block-size **2**Kb

./../source/main --launch-count **10** --memory-type flash --block-size **2**Kb --maximum-buffer