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

Гилев М.А.

Работу проверил:

Ассистент кафедры ВС

Петухова Я.В.

Новосибирск 2021

## Задание

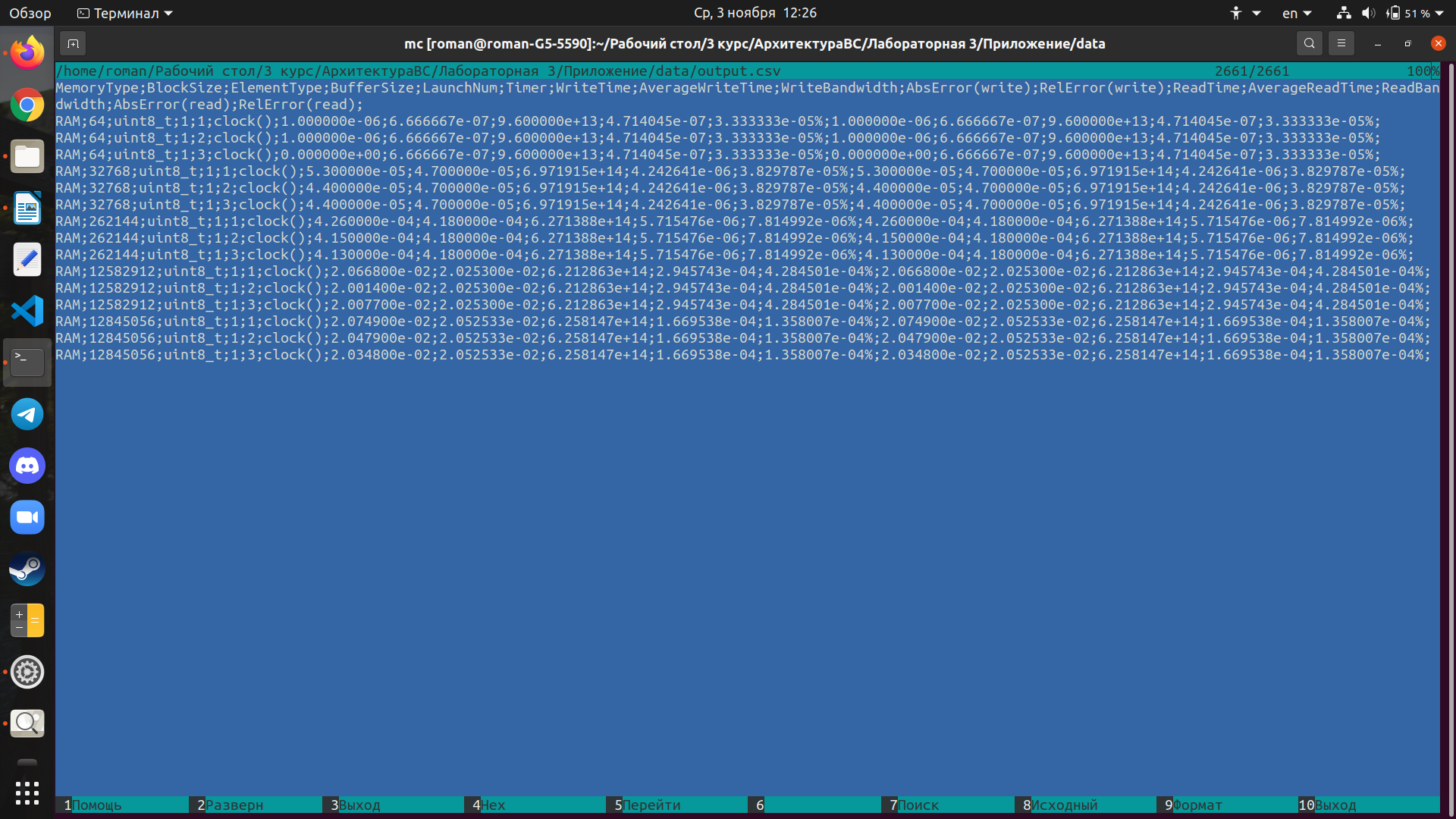
Разработать программу (benchmark) для оценки производительности подсистемы памяти.

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.

Курс «Архитектура вычислительных систем». СибГУТИ. 2020 г.   
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;   
4. \*\* Оценить пропускную способность файла подкачки (windows) или раздела SWAP (linux). Сравнить с   
пропускной способностью RAM, HDD/SSD и flash.

## Результаты работы

Проведена серия испытаний для оценки пропускной способности оперативной памяти. Оценивались блоки данных размером кэш-линии, L1, L2 и L3 (уровни кеша), а также превышающим уровни кеша.

Рис. 1. Оценка кеша

Как видно из Рис. 1., время выполнения (WriteTime) отдельного испытания не превышает 5.3 cекунды (при обработке 32 768 бит)

Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 64 бит — 9.6 Mb/s.

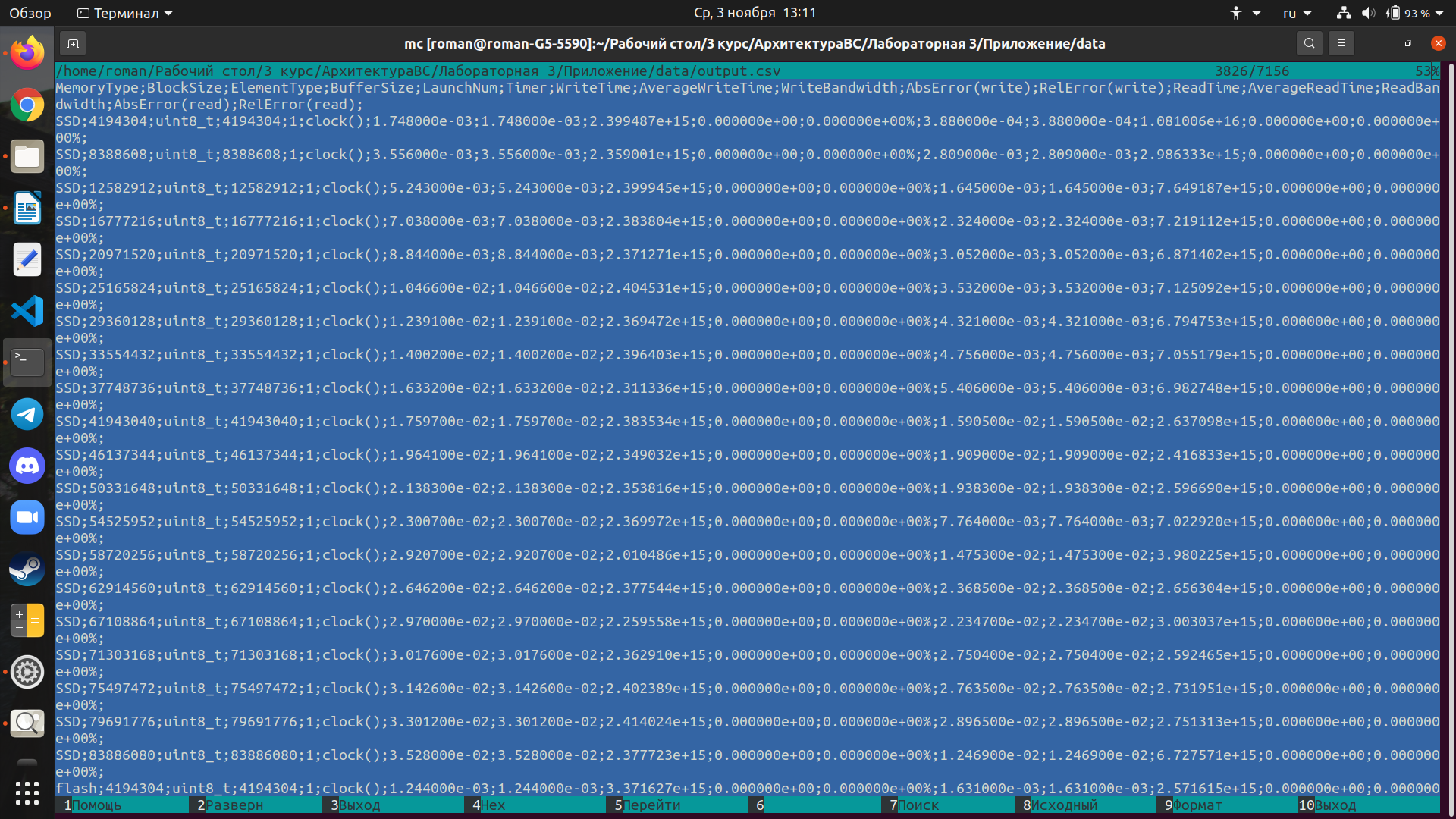
Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 32768 бит — 6.97 Mb/s .

Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 262144 бит — 6.27 Mb/s.

Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 12582912 бит — 6.21 Mb/s.

Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 12845056 бит — 6.26 Mb/s.

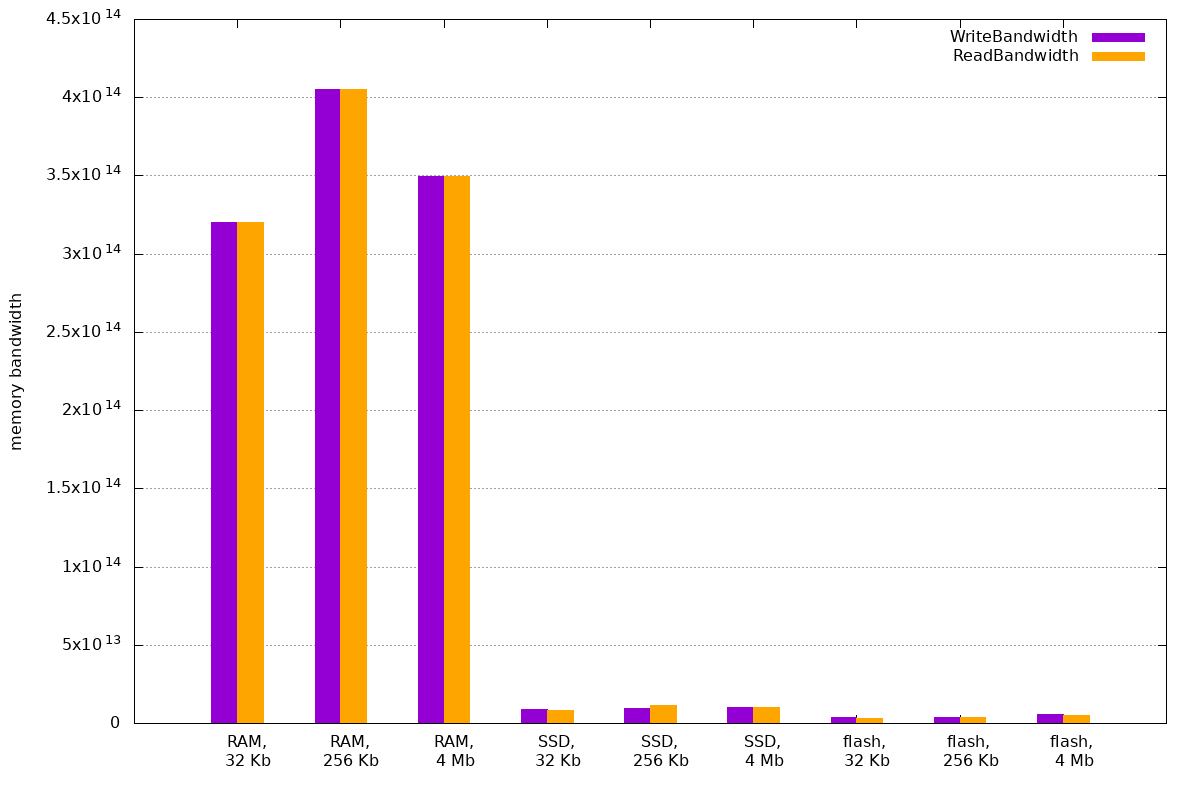
Затем была проведена аналогичная серия из 20 испытаний для SSD и flash соответсвенно. Блоки данных от 4 Мб с шагом 4 Мб.

Рис. 2. Оценка SSD

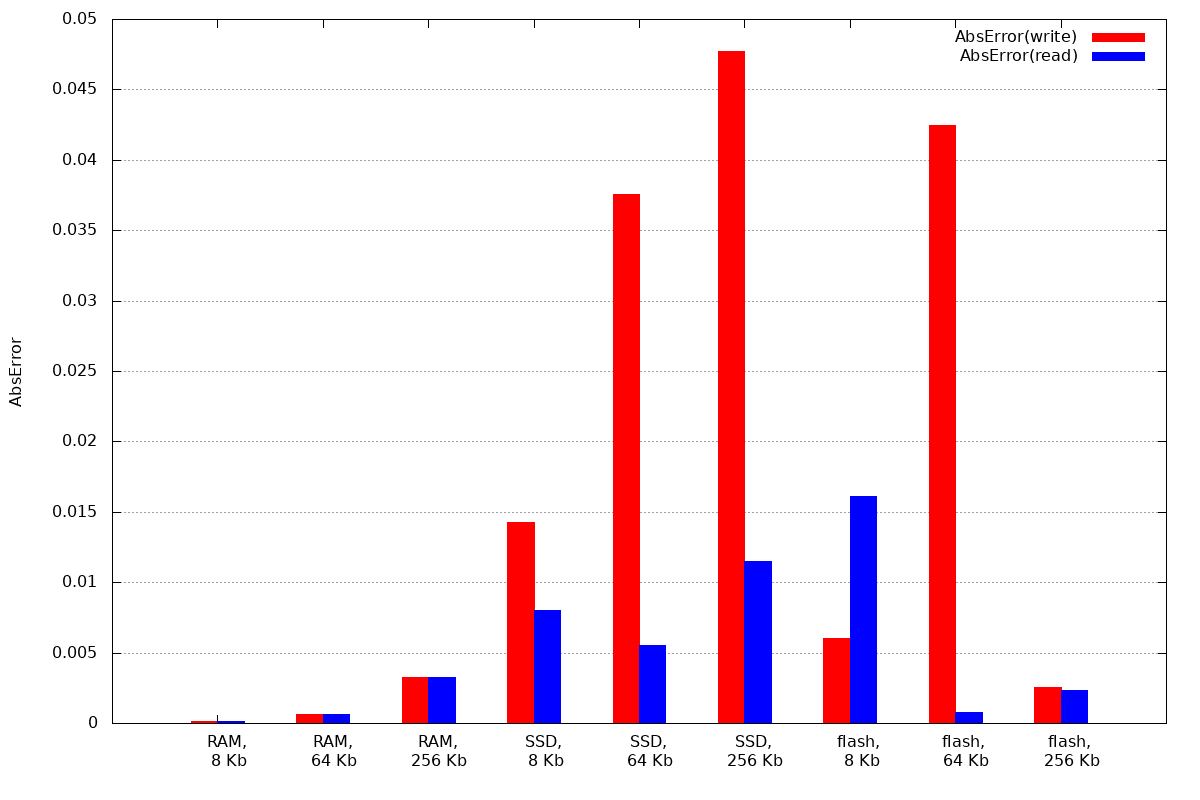
Видим, что:

Пропускная способность оперативной памяти при чтении/записи (Read/Write) блока размером 4194304 бит — Mb/s.

Построены диаграммы.

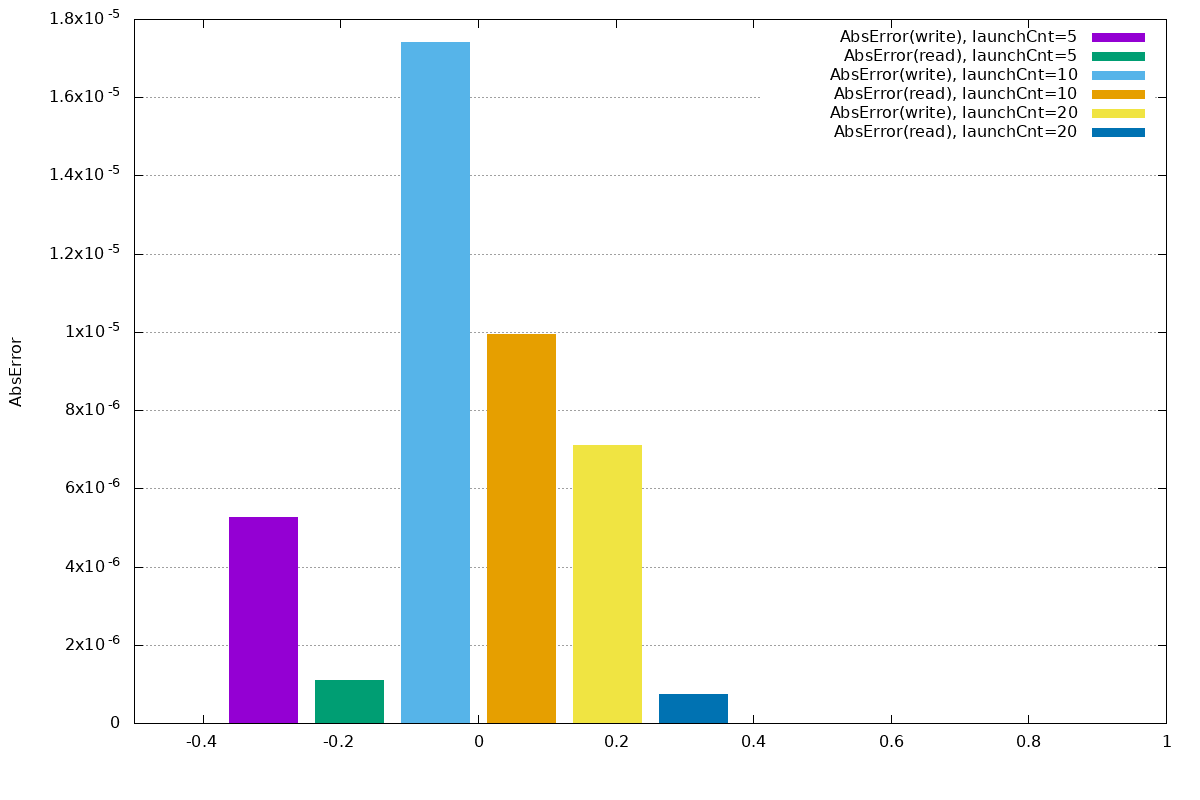
Рис. 3. Write/ReadBandWidth

На Рис. 3. видим пропускную способность (Мб/с) каждого вида памяти в зависимости от объема данных (32 Кб, 256 Кб и 4 Мб). Данный флеш — накопитель имеет физически низкую скорость чтения/записи — результат ожидаемый.

Рис. 4. AbsError

Видим, что абсолютная погрешность измерения времени максимальна при записи блока данных размером 256 Кб на твердотельном накопителе.

Абсолютная погрешность измерения времени достигает значительно больших показателей в ряде испытаний, как с SSD накопителем, так и с flash.

Рис. 5. AbsError(launchCnt)

Приложение

main.cpp

#include <cmath>

#include <cstring>

#include <fstream>

#include <iostream>

#include <string>

#include <sys/time.h>

using namespace std;

double wtime() {

struct timeval t;

gettimeofday(&t, NULL);

return (double) t.tv\_sec + (double) t.tv\_usec \* 1E-6;

}

double getAverage(const double \*array, int size) {

double average = 0;

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

average += array[i];

}

return average / size;

}

long long getSize(string block) {

long long size = 1, \_size;

bool isNum = true;

long long i;

string s = "";

for (i = 0; i < block.size(); i++) {

if (!isdigit(block[i])) {

isNum = false;

break;

}

s.push\_back(block[i]);

}

if (block[i] == 'K' && block[i + 1] == 'b') {

size = 1024;

} else if (block[i] == 'M' && block[i + 1] == 'b') {

size = 1048576;

}

return static\_cast<long long>(stoi(s)) \* size;

}

void initArray(char \*array, long long size) {

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

array[i] = rand() % 256;

}

}

double maxElem(double \*timeBuf, int size) {

double max = timeBuf[0];

for (int i = 1; i < size; i++) {

if (max < timeBuf[i]) {

max = timeBuf[i];

}

}

return max;

}

void writeCSV(int mode, long long blockSize, double \*writeTimeBuf, double avWriteTime, double \*readTimeBuf, double avReadTime, int launchCount) {

string memoryType;

if (mode == 1)

memoryType = "RAM";

else if (mode == 2)

memoryType = "HDD";

else if (mode == 3)

memoryType = "SSD";

else if (mode == 4)

memoryType = "flash";

string elementType = "char";

long long bufferSize = blockSize;

int launchNum = launchCount;

double writeTime = round(writeTimeBuf[launchNum] \* 100) / 100;

string timer = "sys/time.h";

string writeBandWidth = to\_string(round(blockSize / avWriteTime \* 100) / (100 \* 1024 \* 1024));

double absErrorWrite = round((maxElem(writeTimeBuf, launchCount) - avWriteTime) \* 100) / 100;

double relErrorWrite = absErrorWrite / avWriteTime \* 100;

double readTime = round(readTimeBuf[launchNum] \* 100) / 100;

string readBandWidth = to\_string(round(blockSize / avReadTime \* 100) / (100 \* 1024 \* 1024));

double absErrorRead = round((maxElem(readTimeBuf, launchCount) - avReadTime) \* 100) / 100;

double relErrorRead = absErrorRead / avReadTime \* 100;

ofstream fout("file.csv", ios\_base::app);

if (fout.good()) {

fout << memoryType << ";" << blockSize << ";" << elementType << ";" << bufferSize << ";"

<< launchNum << ";" << timer << ";" << writeTime << ";" << avWriteTime << ";"

<< writeBandWidth << ";" << absErrorWrite << ";" << relErrorWrite << "%;"

<< readTime << ";" << avReadTime << ";" << readBandWidth << ";"

<< absErrorRead << ";" << relErrorRead << "%\n";

}

fout.close();

}

double noEchoRead(char \*array, long long size, const string &path = "", int mode = 1) {

double time = -1;

if (mode == 1) {

time = wtime();

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

\*(array + i);

}

time = wtime() - time;

} else {

ifstream fin(path);

if (fin.good()) {

time = wtime();

while (!fin.eof()) {

fin.get();

}

time = wtime() - time;

}

}

return time;

}

pair<double, double> testMemory(long long size, int mode, const string& path = "") {

double timeRead = -1;

double timeWrite = -1;

char \*buffer = new char[size];

if (mode == 1) {

timeWrite = wtime();

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

buffer[i] = static\_cast<char>(rand() % 26 + 65);

}

timeWrite = wtime() - timeWrite;

timeRead = noEchoRead(buffer, size);

} else if (mode > 1 && mode < 5) {

ofstream fout(path);

if (fout.good()) {

initArray(buffer, size);

timeWrite = wtime();

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

fout << buffer[i];

}

timeWrite = wtime() - timeWrite;

}

fout.close();

timeRead = noEchoRead(buffer, size, path, mode);

}

return make\_pair(timeWrite, timeRead);

}

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

srand(time(NULL));

int launchCount = 0;

int mode;

long long size = 0;

string path;

if (argc != 9) {

cout << "./memory\_test –m [RAM/HDD/SSD/flash] –b [size(b/Kb/Mb (b default))] –l [launch count] -p [path]" << endl;

return 1;

} else {

for (int i = 1; i < 9; i += 2) {

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

if (strcmp(argv[i + 1], "RAM") == 0)

mode = 1;

else if (strcmp(argv[i + 1], "HDD") == 0)

mode = 2;

else if (strcmp(argv[i + 1], "SSD") == 0)

mode = 3;

else if (strcmp(argv[i + 1], "flash") == 0)

mode = 4;

} else if (strcmp(argv[i], "-b") == 0) {

string strSize = argv[i + 1];

size = getSize(strSize);

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

launchCount = stoi(argv[i + 1]);

else if (strcmp(argv[i],"-p") == 0 && mode > 1) {

path = argv[i + 1];

path += "/test.txt";

}

}

}

double \*writeTimeBuffer = new double[launchCount];

double \*readTimeBuffer = new double[launchCount];

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

pair<double, double> time;

if (mode != 1) {

time = testMemory(size, mode, path);

}

else {

time = testMemory(size, mode);

}

writeTimeBuffer[i] = time.first;

readTimeBuffer[i] = time.second;

}

double averageWriteTime = getAverage(writeTimeBuffer, launchCount);

double averageReadTime = getAverage(readTimeBuffer, launchCount);

writeCSV(mode, size, writeTimeBuffer, averageWriteTime, readTimeBuffer, averageReadTime, launchCount);

delete(writeTimeBuffer);

delete(readTimeBuffer);

return 0;

}