МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ВТ

Лабораторная работа №2

«Оценка производительности процессора»

Факультет: АВТФ

Группа: АПИМ-25

Выполнил: Бадагов А.В., Клименко К.В.

Преподаватель: Перышкова Е.Н.

Отметка о защите:

Новосибирск 2025

Задание: разработать программу (benchmark) для оценки производительности подсистемы памяти.

1. Написать программу(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) 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) илимодель 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|SSDи flash оценить влияние размерабуфера (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;

4. ** Оценить пропускную способность файла подкачки (windows) или раздела SWAP (linux). Сравнить с пропускной способностью RAM, HDD/SSD и flash.

Ход работы:

Листинг1 часть функции Main

```
    using (var writer = new StreamWriter(outputFile, append: true))
    {
        if (writeHeader)
        {
            writer.WriteLine(
                "MemoryType;BlockSizeBytes;ElementType;BufferSizeBytes;" +
                "LaunchNum;Timer;" +

"WriteTime;AverageWriteTime;WriteBandwidthMBps;AbsErrWrite;RelErrWrite;" +

"ReadTime;AverageReadTime;ReadBandwidthMBps;AbsErrRead;RelErrRead");
        }

        if (memoryType == "RAM")
        {
            BenchmarkRam(
                memoryType,
                blockSizeBytes,
                elementType,
                runs,
                timerName,
                writer);
        }
        else if (isDisk)
        {
            BenchmarkFile(
                memoryType,
                blockSizeBytes,
                elementType,
                runs,
                timerName,
                filePath,
                writer);
        }
    }
```

```
    Console.WriteLine("Done.");
}
```

Функция *Main* организует работу всей программы. На вход она принимает массив строк *args*, который содержит аргументы командной строки, размер блока данных и число запусков испытаний, так же осуществляется проверка на коретность данных и указание место формирования .bin файла.

В листинге 1 программы приведена часть функции *Main* в которой происходит открытие CSV-файла и запуск нужной функции *BenchmarkRam* или *BenchmarkFile* по замеру времени, данные функции будут рассмотрены далее.

new StreamWriter(outputFile, append: true) – открывает текстовый файл для записи.

if (writeHeader) – условие на проверку наличия файла.

Листинг 2 функции, проверка второго параметра block-size

```
static long ParseSize(string s)
{
    s = s.Trim();

    int i = 0;
    while (i < s.Length && char.IsDigit(s[i]))
        i++;

    if (i == 0)
    {
        Console.WriteLine("Неправильно введен размер блока данных");
        Environment.Exit(1);   // полностью завершить программу
    }
    string numPart = s.Substring(0, i);
    if (!long.TryParse(numPart, out long value) || value <= 0)
    {
        Console.WriteLine("Неправильно введен размер блока данных");
        Environment.Exit(1);
    }

    string suffix = s.Substring(i).Trim().ToLowerInvariant();

    if (suffix == "" || suffix == "b")
    {
        return value;
    }
    else if (suffix == "kb")
    {
        return value * 1024;
    }
    else if (suffix == "mb")
    {
        return value * 1024 * 1024;
    }
    else
    {
        Console.WriteLine("Неправильно введен тип блока данных");
        Environment.Exit(1);   // полностью завершить программу
    }
    return 0;
}
```

Листинг 2 организует проверку второго параметра на соответствие размерности Kb, Mb и b, так же что бы значение перед размерностью было целочисленной.

s = s.Trim(); - удаляет пробелы в начале и в конце строки.

while (i < s.Length && char.IsDigit(s[i])) – определяет количество цифр в строке.

Environment.Exit(1) – команда прерывающая работу.

string numPart = s.Substring(0, i); - определения введенного значения размерности блока данных.

suffix = s.Substring(i).Trim().ToLowerInvariant(); – определение типа размерности блока данных.

Листинг 3 часть функции BenchmarkRam, по замеру времени

```
// Запись
for (int r = 0; r < runs; r++)
{
    sw.Restart();
    Buffer.BlockCopy(src, 0, dst, 0, src.Length);
    sw.Stop();
    writeTimes[r] = sw.Elapsed.TotalSeconds;
}

// Чтение
long sum = 0;
for (int r = 0; r < runs; r++)
{
    sw.Restart();
    for (int i = 0; i < dst.Length; i++)
        sum += dst[i];
    sw.Stop();
    readTimes[r] = sw.Elapsed.TotalSeconds;
}
```

Функция *BenchmarkRam* реализует часть задания, связанную с оценкой пропускной способности оперативной памяти при работе с блоками данных фиксированного размера. Создаются два массива *src* и *dst* длиной *blockSizeBytes*. Массив src инициализируется случайными значениями с помощью *Random.NextBytes*.

В листинге 3 представлены два цикла, первый выполняет *Buffer.BlockCopy(src, 0, dst, 0, src.Length)*, копирует весь блок из *src* в *dst.*

sw.Restart() - обнуляет и запускает таймер.

sw.Stop() - останавливает его.

sw.Elapsed.TotalSeconds – длительность испытания в секундах.

Во втором цикле проходится по массиву *dst*, и его элементы суммируются в переменную *sum*. Это обеспечивает чтение данных из памяти. Для каждого прохода измеряется время. Далее в функции запускается функция *ComputeStats* данная функция будет рассмотрена далее.

Листинг 4 часть функции BenchmarkFile, создание данных для замера

```
using (var fs = new FileStream(
    filePath,
    FileMode.Create,
    FileAccess.Write,
    FileShare.None,
    bufferSizeBytes,
    FileOptions.SequentialScan))
{
    long remaining = blockSizeBytes;
    while (remaining > 0)
    {
        int chunk = (int)Math.Min(buffer.Length, remaining);
        fs.Write(buffer, 0, chunk);
        remaining -= chunk;
    }
    fs.Flush(true);
}
```

В программе 4 написан цикл, в котором осуществляется формирование файла, данные которое используются для замеров.

(int)Math.Min(buffer.Length, remaining) – определяется минимальное значение между длиной буфера и количеством значений в сгенерированном массиве.

fs.Write(buffer, 0, chunk) – запись значений из буфера длиной *chunk*.

Листинг 5 часть функции BenchmarkFile, замер времени считывания

```
long sum = 0;
for (int r = 0; r < runs; r++)
{
    using (var fs = new FileStream(
        filePath,
        FileMode.Open,
        FileAccess.Read,
        FileShare.Read,
        bufferSizeBytes,
        FileOptions.SequentialScan))
    {
        sw.Restart();
        int read;
        while ((read = fs.Read(buffer, 0, buffer.Length)) > 0)
        {

            sum += buffer[0];
        }
        sw.Stop();
        readTimes[r] = sw.Elapsed.TotalSeconds;
    }
  }
```

В листинге программы 5 представлена часть основбной программы отвечающей за расчёт времени при считывание данных с носителя.

while ((read = fs.Read(buffer, 0, buffer.Length)) > 0) - в данном цикле реализуется считывание данных и замер времени.

Листинг 6 функции отвечающей за расчёт статистических параметров

```
// статистики
static (double mean, double absErr, double relErr) ComputeStats(double[] times)
{
    int n = times.Length;
    double mean = times.Average();

    double variance = 0.0;
    for (int i = 0; i < n; i++)
    {
        double diff = times[i] - mean;
        variance += diff * diff;
    }
    variance /= n;

    double stdDev = Math.Sqrt(variance);
    double absErr = stdDev / Math.Sqrt(n);
    double relErr = absErr / mean * 100.0 ;

    return (mean, absErr, relErr);
}
```

$$absErr = \frac{\sqrt{\sum_{i=1}^{runs}(t_i - \bar{t})^2}}{runs} \quad (2)$$

$$relErr = \frac{absError}{\bar{t}} 100\% \quad (3)$$

$$taskPerf = \frac{insCount}{\bar{t}} \quad (4)$$

где $\bar{t}$ – среднее значение времени выполнения перемножения матрицы.

2 Оценим пропускную способность оперативной памяти при работе с блоками данных равными объёму кэш-линии, кэш-памяти L1, L2 и L3 уровня и превышающего его см. рисунок 1. Для HDD|SSD и flash проведем серию из 20 испытаний с блоками данных начиная с 4 Мб с шагом 4Мб см. рисунок 2. Для HDD|SSDи flash оценить влияние размера буфера (BufferSize) на пропускную способность памяти.

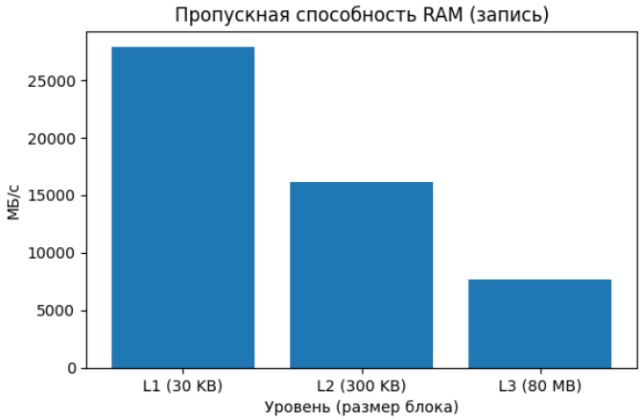

Рисунок 1 – Оценка пропускной способности оперативной памяти, при записи

В соответсвии с рисунком 1 можно сделать заключение, что с увеличением объёмом кэш-линии пропускная способность уменьшается, L2 меньше L1 в 1.72, а L3 меньше L1 в 3.63 раза. L1= 27901.79 Мб/с, L2 = 16186.12 Мб/с, L3 = 7685,354 Мб/с.

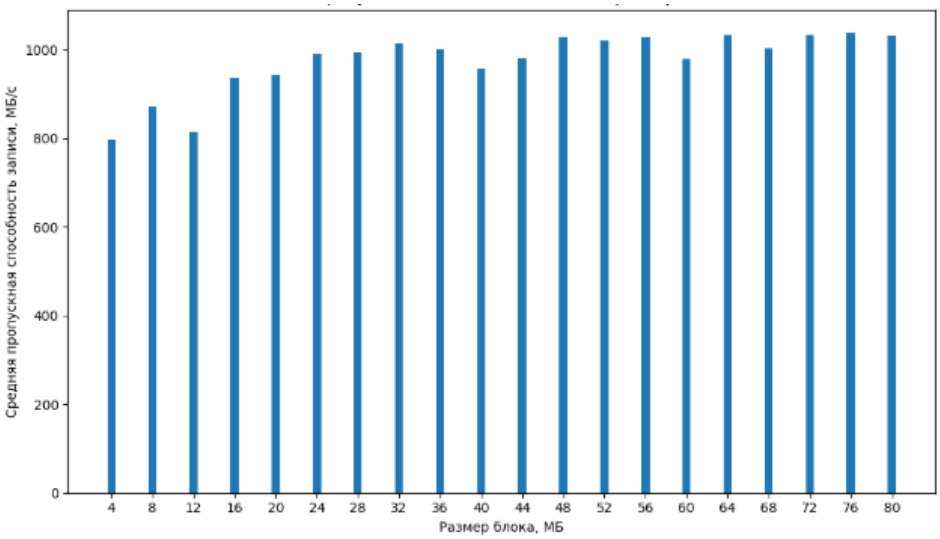

Рисунок 2 - Оценка пропускной способности SSD

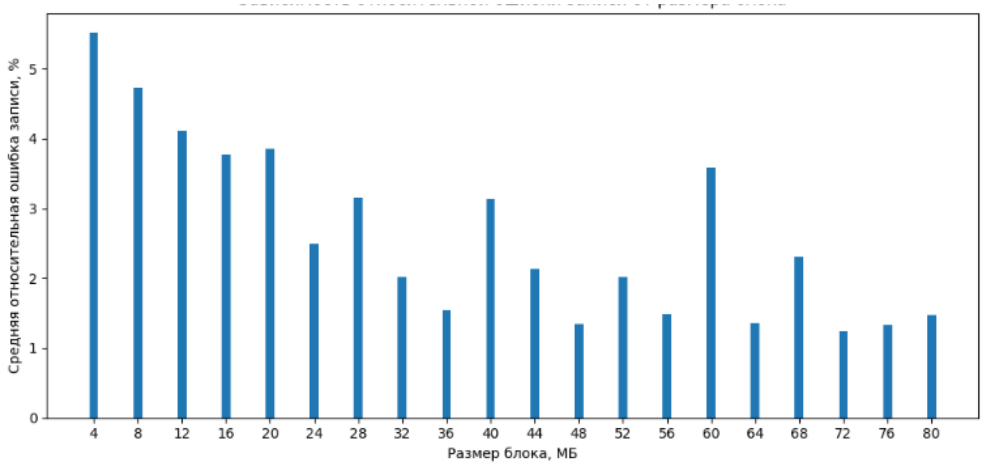

Рисунок 3 – Изменение относительной погрешности

В соответсии с рисунком 2 можно сделать заключение, что с увеличением блока данных пропускная способность имеет тенденцию увеличения, в то время как относительная ошибка имеет противоположный характер см. рисунок 3.

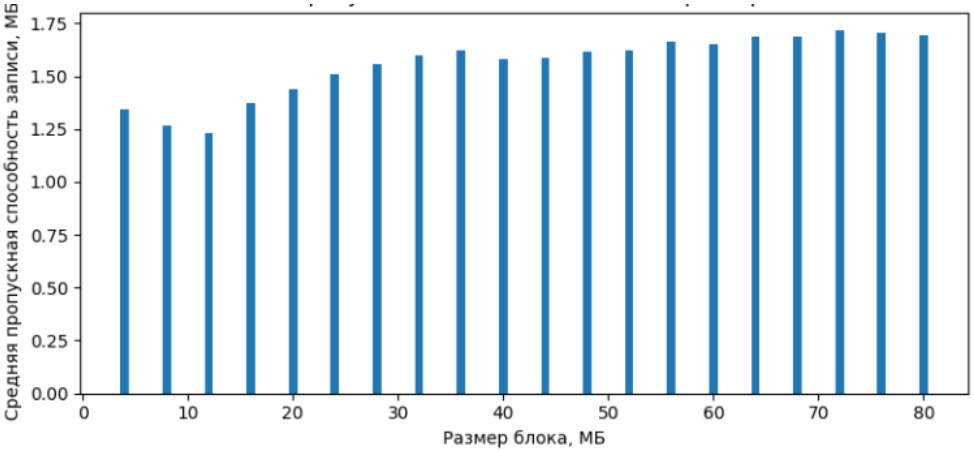

Рисунок 4 - Оценка пропускной способности FLASH записи

В соответвсии с рисунком 4 можно сделать заключение, что с увеличением объёмом блока данных пропускная способность увеличивается на 27%. Малые блоки — заметно медленнее, блоки ≥ 24–32 МБ — выход на плато, дальше увеличения размера блока почти не дают выигрыша, только мелкие колебания из-за внутренних особенностей контроллера и ОС.

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;

Зависимость пропускной способности при записи для разных типов представлены с 1 по 4 рисунок.

Зависимость пропускной способности при чтении для разных типов представлены с 5 по 7 рисунок

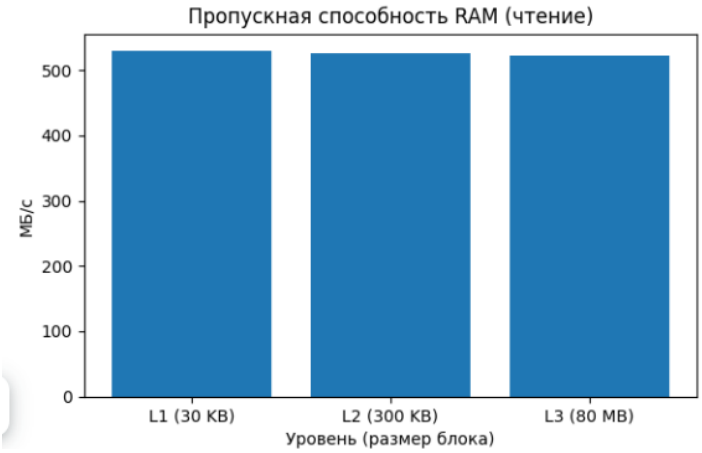

Рисунок 5 - Оценка пропускной способности оперативной памяти, при чтении (L1 = 529.4 Мб/с, L2 = 526.2 Мб/с, L3 = 521.4 Мб/с)

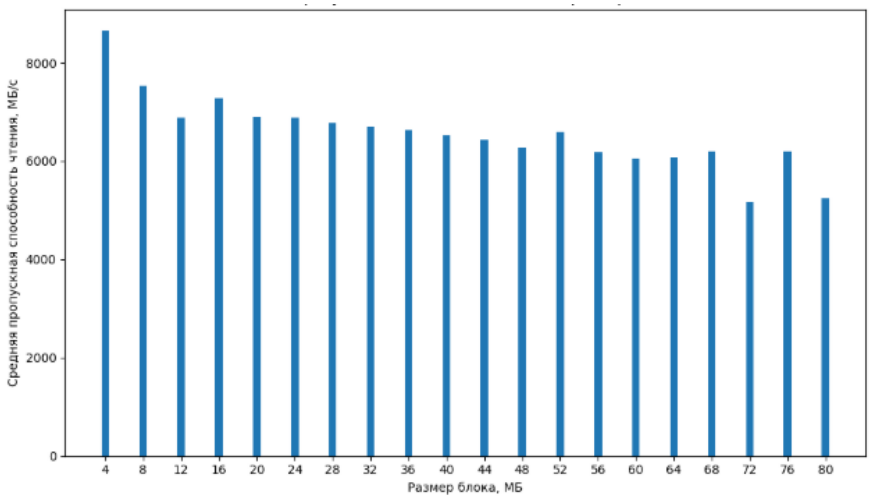

Рисунок 6 - Оценка пропускной способности SSD, при чтении

Рисунок 7 - Оценка пропускной способности FLASH, при чтении

В соответчики с полученными значениями при чтении пропускная способность падает при увеличении размера блока для каждого типа памяти.

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

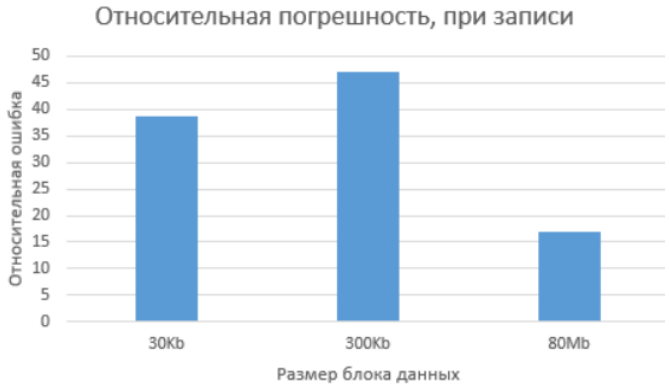

Рисунок 8 – Значение относительной погрешности при изменении блока данных RAM

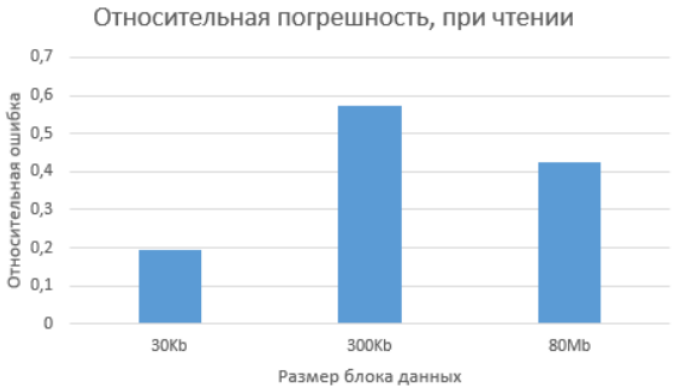

Рисунок 9 – Значение относительной погрешности при изменении блока данных RAM

Рисунок 10 – Значение относительной погрешности при изменении блока данных SSD

Рисунок 11 – Значение относительной погрешности при изменении блока данных SSD

Запись: при увеличении блока относительная ошибка уменьшается с 5.5 % до 1.3 % в 4раза, общий тренд графика направлен на уменьшение.

Чтение: ошибка снижается с 4.6 % до 1 %, при этом присутвуют выбросы при которых ошибка достигает 2% - 2.1%. Значение ошибки более крупных блоков дает более стабильные замеры**,** чем мелкие 4–8 МБ.

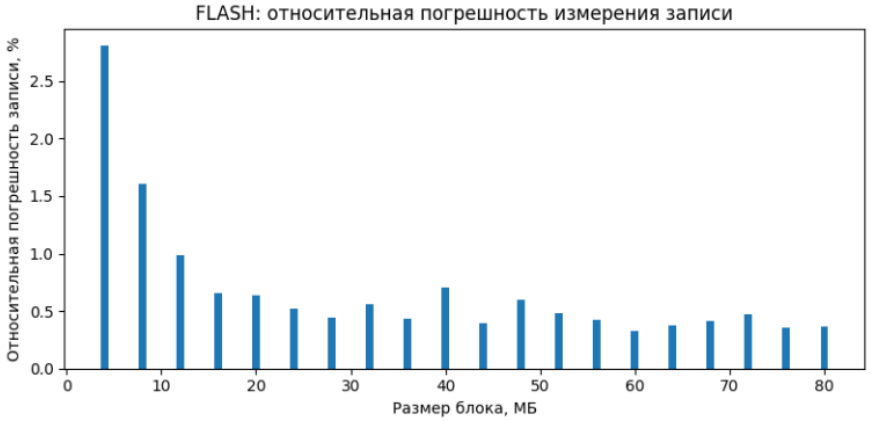

Рисунок 12 – Значение относительной погрешности при изменении блока данных FLASH

Рисунок 13 – Значение относительной погрешности при изменении блока данных FLASH

Относительная погрешность измерения у Flash значительно уменьшается с ростом блока: с 2.8%, 4МБ, до 0.3–0.5%, блоки ≥ 20МБ. Это отражает переход от режима, где результат доминирует системный шум, к режиму, где измеряем стабильную стационарную скорость накопителя

3) Зависимость погрешности измерений от числа испытаний LaunchNum;

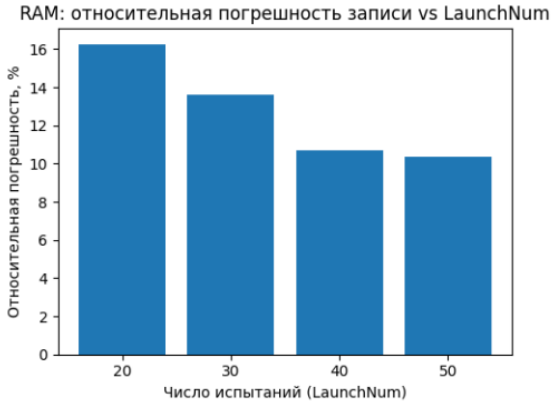

Рисунок 14 - Значение относительной погрешности при изменении количества испытаний RAM

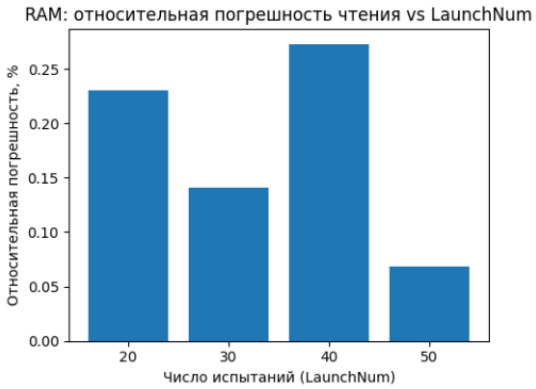

Рисунок 15 - Значение относительной погрешности при изменении количества испытаний RAM

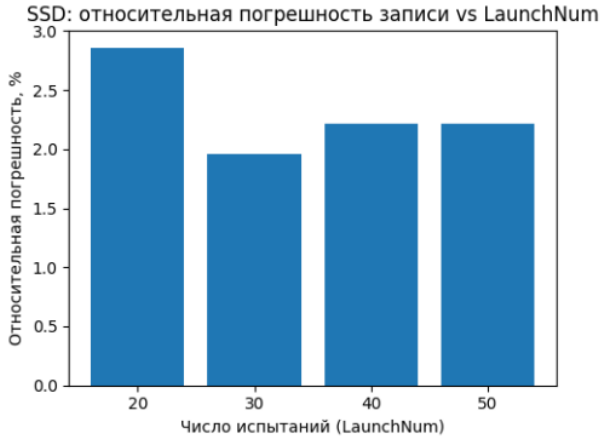

Рисунок 16 - Значение относительной погрешности при изменении количества испытаний SSD

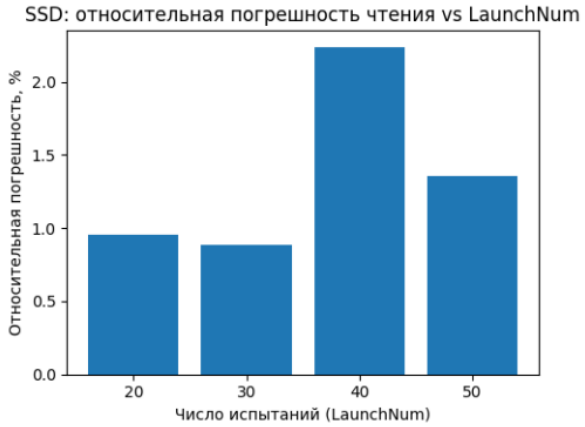

Рисунок 17 - Значение относительной погрешности при изменении количества испытаний SSD

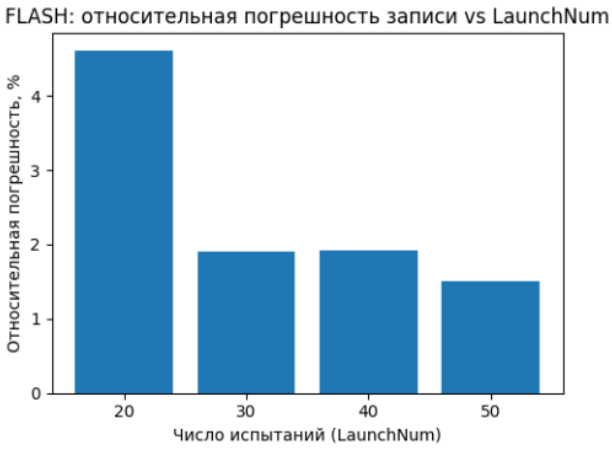

Рисунок 18 - Значение относительной погрешности при изменении количества испытаний FLASH

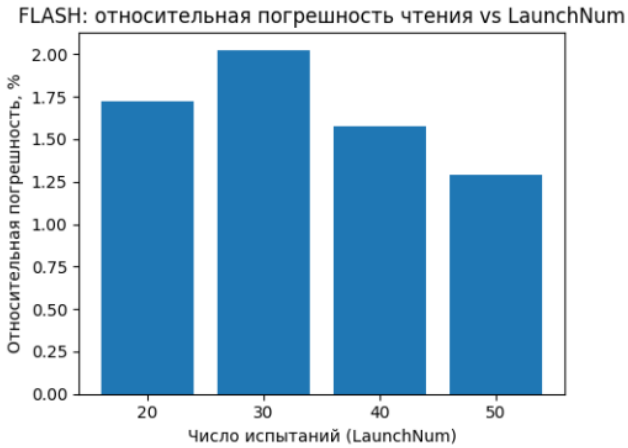

Рисунок 19 - Значение относительной погрешности при изменении количества испытаний FLASH

Минимальная погрешность при максимальном числе испытаний получается у FLASH (1.5 %), затем SSD (2.2 %), и наихудшая по относительной ошибке RAM (10 %) записи.

При чтении результаты противоположные лучший по стабильности RAM погрешность маленькая, доли процента, SSD и FLASH близки друг к другу, с ошибкой порядка 1–2 %, и слабой, но заметной пользой от увеличения числа испытаний.

Увеличение числа испытаний всегда помогает, но эффект зависит от скорости устройства:

- для RAM уменьшение ошибки записи заметно только до 40 запусков;
- для FLASH рост LaunchNum с 20 до 50 даёт очень сильное улучшение по записи;
- для SSD картина менее гладкая, так как на него сильнее влияют фоновые процессы.

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

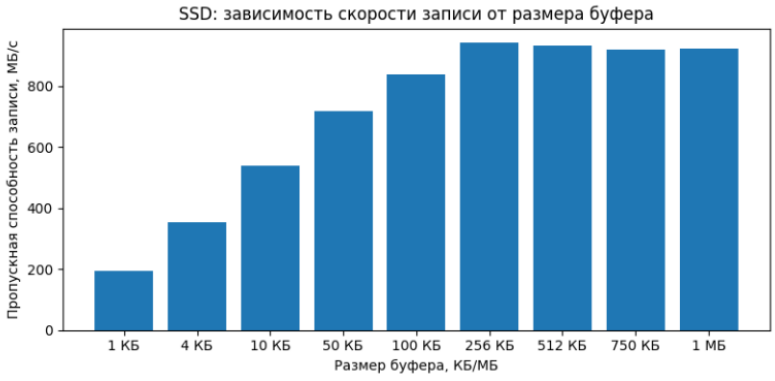

Рисунок 19 - Значение пропускной способности при изменении размера буфера SSD

До 100–256 КБ пропускная способность растёт линейно. В районе 256 КБ достигается постоянное значение около 0.9–0.95 ГБ/с. Дальнейшее увеличение буфера почти не даёт выигрыша: ограничивает уже сам SSD и интерфейс, а не размер буфера.

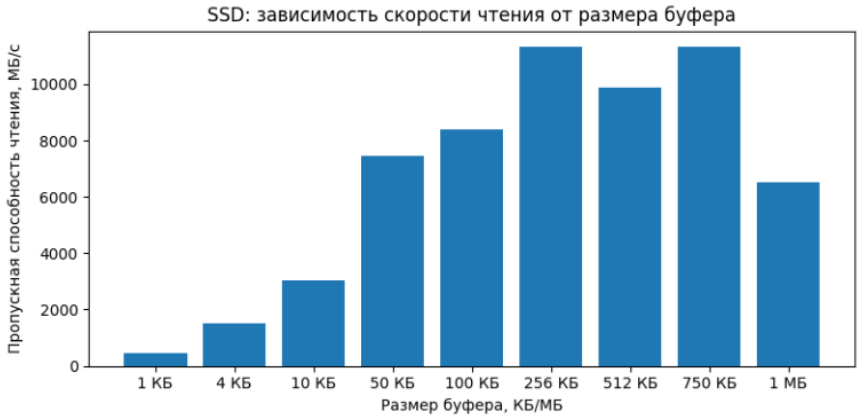

Рисунок 20 - Значение пропускной способности при изменении размера буфера SSD

Аналогично записи, рост очень резкий от 1 КБ до 50–100 КБ: маленькие буферы тратят много времени системный вызов, переключение контекста, управление кэшем файловой системы. В диапазоне 100–750 КБ достигается высокий уровень 8–11 ГБ/с. Падение при 1 МБ связано с тем, что часть больших чтений уже не помещается целиком в файловый кэш и сильнее зависит от реальной скорости SSD и текущей нагрузки системы.

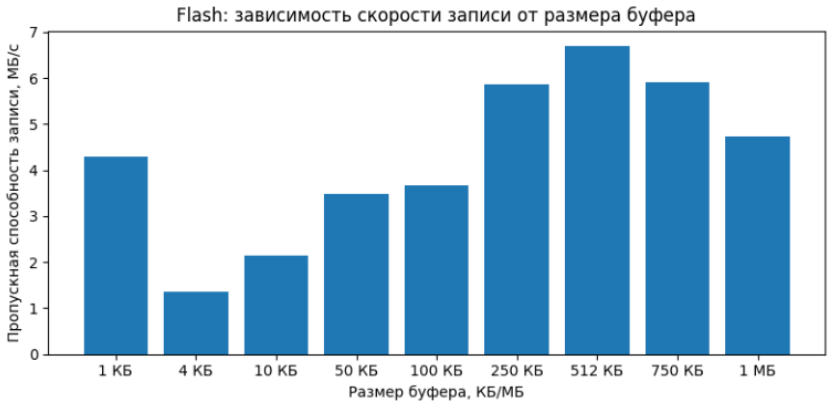

Рисунок 21 - Значение пропускной способности при изменении размера буфера FLASH

Увеличение буфера с 1 КБ до 250–512 КБ приводит к росту пропускной способности с 2–4 МБ/с до ~6–7 МБ/с: контроллеру выгодно получать данные крупными последовательными порциями. При увеличении до 750 КБ–1 МБ скорость немного падает, т.к. сложнее вписать такие запросы в свои внутренние блоки, поэтому растут задержки и время внутренних операций.

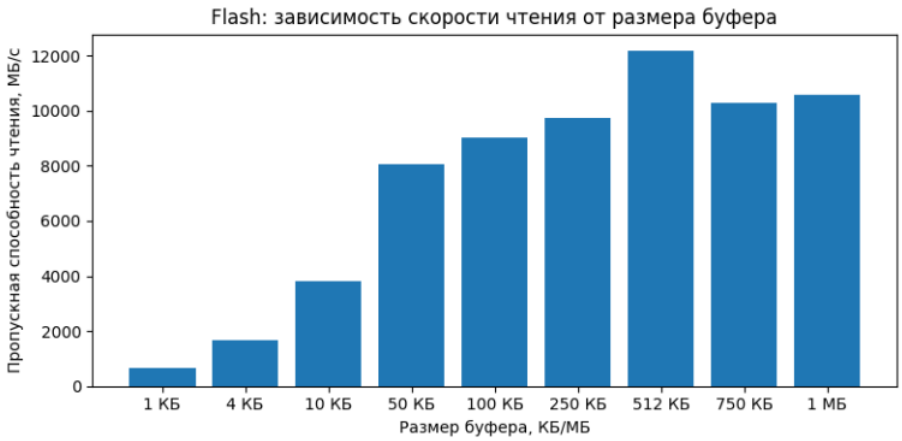

Рисунок 22 - Значение пропускной способности при изменении размера буфера FLASH

Результат похож на SSD: при малых буферах чтение сильно страдает; начиная с 50–100 КБ выходит на плато, но ещё сильнее зависит от кэша ОС; физическая пропускная способность флешки скрыта за очень быстрым повторным чтением из RAM.

Вывод: в ходе работы была разработана и реализована на языке C# benchmark-программа для оценки пропускной способности подсистемы памяти (RAM, SSD и flash). Программа принимает тип памяти, размер блока и число испытаний, переводит размер из байт/КБ/МБ в байты, выполняет серию операций последовательной записи и чтения массива или файла, измеряет время с помощью Stopwatch и для каждого запуска формирует строку в CSV с оценками средней скорости, абсолютной и относительной погрешности. На основе полученных данных построены зависимости пропускной способности и погрешности от размера блока, количества запусков и размера буфера. Показано, что для RAM при переходе от объёма кэш-линии к уровням L2 и L3 пропускная способность записи падает примерно в 3,5 раза, тогда как скорость чтения почти не меняется из-за доминирования накладных расходов цикла. Для SSD и flash рост блока от 4 до 80 МБ приводит к увеличению пропускной способности записи с выходом на плато, а относительная погрешность в этих условиях уменьшается в несколько раз. Анализ влияния размера буфера показал, что очень малые буферы (1–10 КБ) сильно снижают скорость, тогда как диапазон 50–256 КБ близок к оптимальному. В целом полученные результаты демонстрируют, что производительность подсистемы памяти существенно зависит не только от типа носителя, но и от выбранных параметров доступа.

Приложение А.

Листинг всей программы на C#

```
using System;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Linq;

class Program
{
        static void Main(string[] args)
        {
        if (args.Length != 3){
            Console.WriteLine("Недостаточно аргументов.");
            Console.WriteLine("Обязательные параметры:");
            Console.WriteLine("  1) MemoryType  (RAM | HDD | SSD | FLASH)");
            Console.WriteLine("  2) BlockSize   (Размер блока данных в байтах, Кб
или Мб)");
            Console.WriteLine("  3) Runs        (целое число запусков)");

            if (args.Length == 0)
            {
                Console.WriteLine("Вы не указали ни одного параметра.");
            }
            else if (args.Length == 1)
            {
                Console.WriteLine($"Указан только MemoryType = {args[0]}, но не
заданы BlockSize и Runs.");
            }
            else if (args.Length == 2)
            {
                Console.WriteLine($"Указаны MemoryType = {args[0]} и BlockSize =
{args[1]}, но не задан Runs.");
            }
            return;
        }


            string memoryType = args[0].ToUpperInvariant();
            string blockSizeStr = args[1];
            string runsStr = args[2];


            if (!int.TryParse(runsStr, out int runs) || runs <= 0)
            {
                Console.WriteLine("Runs должен быть положительным целым числом");
                return;
            }

            long blockSizeBytes = ParseSize(blockSizeStr);

            bool isDisk =
                memoryType == "HDD" ||
                memoryType == "SSD" ||
                memoryType == "FLASH";

            if (isDisk)
            {
                long minBlock = 4L * 1024 * 1024;
                if (blockSizeBytes < minBlock)
```

```
                {
                    blockSizeBytes = minBlock;
                }

                if (runs < 20)
                {
                   runs = 20;
                }
            }
        string filePath;

        if (memoryType == "HDD" || memoryType == "SSD")
        {
            filePath = @"C:\lab3\memory_test.bin";
        }
        else
        {
            filePath = @"F:\lab3\memory_test.bin";
        }
        string outputFile = "memory_results.csv";
            string timerName = "Stopwatch";
            string elementType = "byte";

            Console.WriteLine("Memory benchmark");
            Console.WriteLine($"MemoryType: {memoryType}");
            Console.WriteLine($"BlockSize:  {blockSizeBytes} bytes");
            Console.WriteLine($"Runs:       {runs}");
            Console.WriteLine($"Output ->   {outputFile}");

            bool writeHeader = !File.Exists(outputFile);

        using (var writer = new StreamWriter(outputFile, append: true))
        {
            if (writeHeader)
            {
                writer.WriteLine(
                    "MemoryType;BlockSizeBytes;ElementType;BufferSizeBytes;" +
                    "LaunchNum;Timer;" +

"WriteTime;AverageWriteTime;WriteBandwidthMBps;AbsErrWrite;RelErrWrite;" +

"ReadTime;AverageReadTime;ReadBandwidthMBps;AbsErrRead;RelErrRead");
            }

            if (memoryType == "RAM")
            {
                BenchmarkRam(
                    memoryType,
                    blockSizeBytes,
                    elementType,
                    runs,
                    timerName,
                    writer);
            }
            else if (isDisk)
            {
                BenchmarkFile(
                    memoryType,
                    blockSizeBytes,
                    elementType,
                    runs,
```

```
                    timerName,
                    filePath,
                    writer);
            }
        }

        Console.WriteLine("Done.");
    }

    // Проверка второго аргумента
    static long ParseSize(string s)
    {
        s = s.Trim();

        int i = 0;
        while (i < s.Length && char.IsDigit(s[i]))
            i++;

        if (i == 0)
        {
            Console.WriteLine("Неправильно введен размер блока данных");
            Environment.Exit(1);   // полностью завершить программу
        }
        string numPart = s.Substring(0, i);
        if (!long.TryParse(numPart, out long value) || value <= 0)
        {
            Console.WriteLine("Неправильно введен размер блока данных");
            Environment.Exit(1);
        }


        string suffix = s.Substring(i).Trim().ToLowerInvariant();



        if (suffix == "" || suffix == "b")
        {
            return value;
        }
        else if (suffix == "kb")
        {
            return value * 1024;
        }
        else if (suffix == "mb")
        {
            return value * 1024 * 1024;
        }

        else
        {
            Console.WriteLine("Неправильно введен тип блока данных");
            Environment.Exit(1);   // полностью завершить программу
        }
        return 0;
    }

    // RAM
    static void BenchmarkRam(
        string memoryType,
        long blockSizeBytes,
        string elementType,
```

```
        int runs,
        string timerName,
        StreamWriter writer)
    {
        // Размер буфера = размер блока (для RAM)
        long bufferSizeBytes = blockSizeBytes;

        byte[] src = new byte[blockSizeBytes];
        byte[] dst = new byte[blockSizeBytes];

        // Инициализируем случайными данными
        var rnd = new Random(123);
        rnd.NextBytes(src);

        double[] writeTimes = new double[runs];
        double[] readTimes = new double[runs];

        var sw = new Stopwatch();

        // Запись
        for (int r = 0; r < runs; r++)
        {
            sw.Restart();
            Buffer.BlockCopy(src, 0, dst, 0, src.Length);
            sw.Stop();
            writeTimes[r] = sw.Elapsed.TotalSeconds;
        }

        // Чтение
        long sum = 0;
        for (int r = 0; r < runs; r++)
        {
            sw.Restart();
            for (int i = 0; i < dst.Length; i++)
                sum += dst[i];
            sw.Stop();
            readTimes[r] = sw.Elapsed.TotalSeconds;
        }

        var (meanW, absErrW, relErrW) = ComputeStats(writeTimes);
        var (meanR, absErrR, relErrR) = ComputeStats(readTimes);

        double writeBandwidthMBps = (blockSizeBytes / meanW) / (1024*1024);
        double readBandwidthMBps = (blockSizeBytes / meanR) / (1024*1024);

        // Запись в CSV
        for (int r=0; r<runs; r++)
        {
            int launchNum = r+1;

            writer.WriteLine(string.Join(";",
                memoryType,
                blockSizeBytes,
                elementType,
                bufferSizeBytes,
                launchNum,
                timerName,
                writeTimes[r],
                meanW,
                writeBandwidthMBps,
                absErrW,
```

```
                relErrW,
                readTimes[r],
                meanR,
                readBandwidthMBps,
                absErrR,
                relErrR
            ));
        }
    }

    // HDD/SSD/FLASH
    static void BenchmarkFile(
        string memoryType,
        long blockSizeBytes,
        string elementType,
        int runs,
        string timerName,
        string filePath,
        StreamWriter writer)
    {

        int bufferSizeBytes = 4*1024;


        byte[] buffer = new byte[bufferSizeBytes];
        var rnd = new Random(456);
        rnd.NextBytes(buffer);

        double[] writeTimes = new double[runs];
        double[] readTimes = new double[runs];


        var sw = new Stopwatch();

        // Измерение записи
        for (int r = 0; r < runs; r++)
        {
            using (var fs = new FileStream(
                filePath,
                FileMode.Create,
                FileAccess.Write,
                FileShare.None,
                bufferSizeBytes,
                FileOptions.SequentialScan))
            {
                long remaining = blockSizeBytes;

                sw.Restart();
                while (remaining > 0)
                {
                    int chunk = (int)Math.Min(buffer.Length, remaining);
                    fs.Write(buffer, 0, chunk);
                    remaining -= chunk;
                }
                fs.Flush(true);
                sw.Stop();

                writeTimes[r] = sw.Elapsed.TotalSeconds;
            }
        }
```

```
        //  Измерение чтения
        long sum = 0;
        for (int r = 0; r < runs; r++)
        {
            using (var fs = new FileStream(
                filePath,
                FileMode.Open,
                FileAccess.Read,
                FileShare.Read,
                bufferSizeBytes,
                FileOptions.SequentialScan))
            {
                sw.Restart();
                int read;
                while ((read = fs.Read(buffer, 0, buffer.Length)) > 0)
                {

                    sum += buffer[0];
                }
                sw.Stop();
                readTimes[r] = sw.Elapsed.TotalSeconds;
            }
        }

        var (meanW, absErrW, relErrW) = ComputeStats(writeTimes);
        var (meanR, absErrR, relErrR) = ComputeStats(readTimes);

        double writeBandwidthMBps = (blockSizeBytes / meanW) / (1024*1024);
        double readBandwidthMBps = (blockSizeBytes / meanR) / (1024*1024);

        long bufferSizeForCsv = bufferSizeBytes;

        for (int r = 0; r < runs; r++)
        {
            int launchNum = r + 1;

            writer.WriteLine(string.Join(";",
                memoryType,
                blockSizeBytes,
                elementType,
                bufferSizeForCsv,
                launchNum,
                timerName,
                writeTimes[r],
                meanW,
                writeBandwidthMBps,
                absErrW,
                relErrW,
                readTimes[r],
                meanR,
                readBandwidthMBps,
                absErrR,
                relErrR
            ));
        }
    }

    // статистики
    static (double mean, double absErr, double relErr) ComputeStats(double[] times)
```

```
    {
        int n = times.Length;
        double mean = times.Average();

        double variance = 0.0;
        for (int i = 0; i < n; i++)
        {
            double diff = times[i] - mean;
            variance += diff * diff;
        }
        variance /= n;

        double stdDev = Math.Sqrt(variance);
        double absErr = stdDev / Math.Sqrt(n);
        double relErr = absErr / mean * 100.0 ;

        return (mean, absErr, relErr);
    }

}
```