###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Влияние кэш-памяти на время обработки массивов»

студента 2 курса, группы 20204

**Дронова Дениса Юрьевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2021

**СОДЕРЖАНИЕ**

[ЦЕЛИ 3](#_Toc89469519)

[ЗАДАНИЕ 3](#_Toc89469520)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc89469521)

[Приложение 1. *Листинг программы* 6](#_Toc89469522)

# ЦЕЛИ

1. Исследование зависимости времени доступа к данным в памяти от их объема
2. Исследование зависимости времени доступа к данным в памяти от порядка их обхода.

# ЗАДАНИЕ

1. Написать программу, многократно выполняющую обход массива заданного размера тремя способами
2. Для каждого размера массива и способа обхода измерить среднее время доступа к одному элементу (в тактах процессора). Построить графики зависимости среднего времени доступа от размера массива
3. На основе анализа полученных графиков:

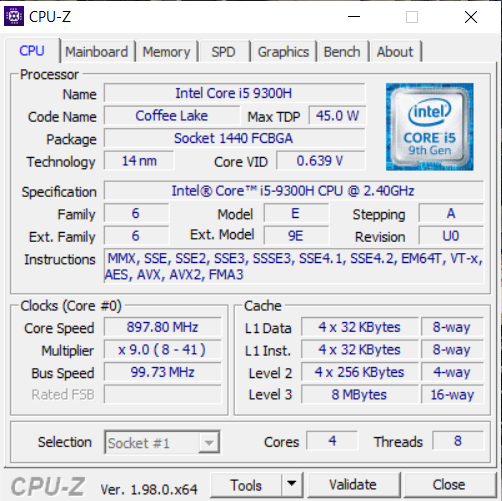
• определить размеры кэш-памяти различных уровней, обосновать ответ, сопоставить результат с известными реальными значениями;

• определить размеры массива, при которых время доступа к элементу массива при случайном обходе больше, чем при прямом или обратном; объяснить причины этой разницы во временах.

# ОПИСАНИЕ РАБОТЫ

В ходе выполнения практической работы было произведено три варианта заполнения массива: последовательное(прямое), обратное, случайное. Все варианты массивов были протестированы в цикле на время обращения к элементам (при помощи конструкции k = arr[k]). Каждый вариант проходил данный тест 100 раз для более точного определения среднего времени. Предварительно измерив размеры кэша разных уровней для конкретного процессора при помощи утилиты CPU-Z, были выбраны границы размеров массива: минимальный порог - 256 элементов int (1Kb – помещается в кэш L1), максимальный порог – 8388608 int’ов (32 Mb – превышает размера кэша L3). По результатам замеров были получены значения числа тактов для каждого варианта обхода массива (прямой, обратный, случайный), для каждого размера массива в вышеуказанном диапазоне (размер массива увеличивался на каждой итерации в 1,2 раза). На основе полученных значений числа тактов были построены графики, приведенные ниже.

Строка компиляции: g++ -O1 CACHE.cpp -o CACHE.exe

**

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения практической работы были исследованы зависимости времени доступа к данным в памяти от их объема, а также от порядка их обхода. На основе результатов замеров можно сделать вывод о том, что время доступа при прямом и обратном обходах сохраняется неизменным независимо от размера массива, так как осуществляется аппаратная предвыборка данных. Однако при случайном обходе время доступа к данным увеличивается пропорционально размеру массива (с увеличением доли массива, не помещающейся в определенные уровни кэша).

# Приложение 1. *Листинг программы*

#include <iostream>

#include <intrin.h>

#include <cmath>

#include <ctime>

void DirectFill(int \*arr, int size)

{

    for (int i = 0; i < size - 1; ++i)

    {

        arr[i] = i + 1;

    }

    arr[size - 1] = 0;

}

void ReversedFill(int \*arr, int size)

{

    for (int i = size - 1; i > 0; --i)

    {

        arr[i] = i - 1;

    }

    arr[0] = size - 1;

}

int RandFromRange(int min, int max)

{

    return (rand() % (max - min + 1) + min);

}

void RandomFill(int \*array, size\_t size)

{

    for (size\_t i = 0; i < size; i++)

    {

        array[i] = i;

    }

    for (size\_t i = size - 1; i >= 1; i--)

    {

        int j = RandFromRange(0, i - 1);

        std::swap(array[i], array[j]);

    }

    bool \*used = new bool[size];

    for (size\_t i = 0; i < size; i++)

    {

        used[i] = false;

    }

    bool is\_permutation\_found = false;

    int lastIdx = -1;

    while (!is\_permutation\_found)

    {

        int i = 0;

        while (i < size && used[i])

        {

            i++;

        }

        if (i == size)

        {

            array[lastIdx] = 0;

            is\_permutation\_found = true;

        }

        else

        {

            if (lastIdx != -1)

            {

                array[lastIdx] = i;

            }

            while (!used[i])

            {

                used[i] = true;

                lastIdx = i;

                i = array[i];

            }

        }

    }

    delete[] used;

}

double CountTime(int \*arr, int size)

{

    //CACHE PREHEAT

    int k = 0;

    for (int i = 0; i < size; ++i)

    {

        k = arr[k];

    }

    if (k == 585)

        std::cout << "WOOOW!" << std::endl;

    double time = UINT64\_MAX;

    for (int j = 0; j < 100; ++j)

    {

        double start = \_\_rdtsc();

        for (int i = 0, k = 0; i < size; ++i)

        {

            k = arr[k];

            if (k == (size + 1))

                std::cout << "WOOOW!" << std::endl;

        }

        double end = \_\_rdtsc();

        if (end - start < time)

        {

            time = end - start;

        }

    }

    return (time / size);

}

int main()

{

    //256 - 1Kb

    //8388608 - 32Mb

    for (int N = 256; N < 8388608; N \*= 1.2)

    {

        std::cout << "Size: " << N << std::endl;

        int \*arr = new int[N];

        DirectFill(arr, N);

        double time = CountTime(arr, N);

        std::cout << "Direct bypass: " << time << " tacts" << std::endl;

        ReversedFill(arr, N);

        time = CountTime(arr, N);

        std::cout << "Reverse bypass: " << time << " tacts" << std::endl;

        RandomFill(arr, N);

        time = CountTime(arr, N);

        std::cout << "Random bypass: " << time << " tacts" << std::endl;

        std::cout << std::endl

                  << std::endl;

        delete[] arr;

    }

    return 0;

}