###### 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 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 2](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 4](#__RefHeading___4)

[Приложение 1 4](#__RefHeading___5)

[7](#__RefHeading___6)

[Приложение 2 7](#__RefHeading___7)

# ЦЕЛЬ

* Исследование зависимости времени доступа к данным в памяти от их объема.
* 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:

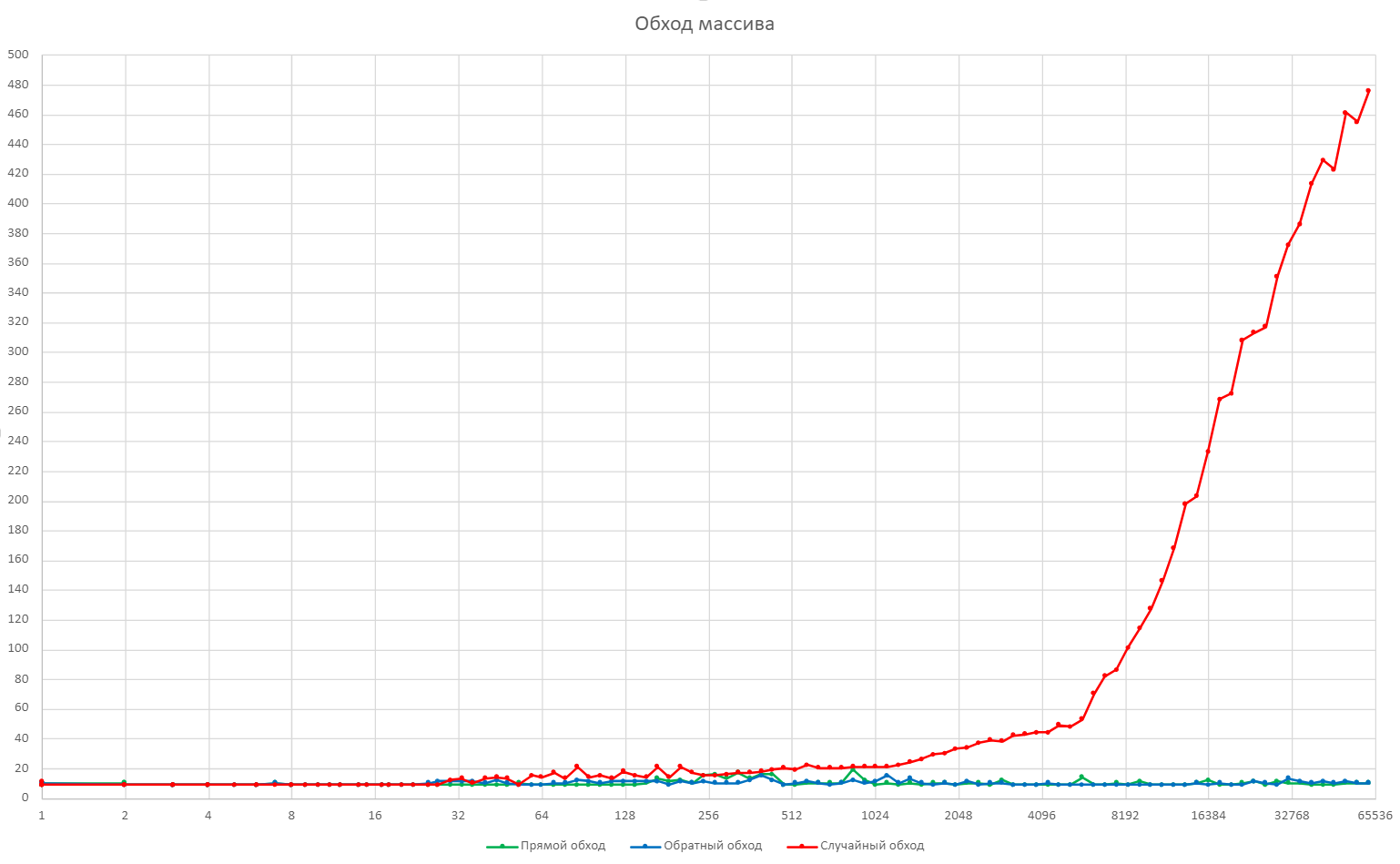
* определить размеры кэш-памяти различных уровней, обосновать ответ, сопоставить результат с известными реальными значениями;
* определить размеры массива, при которых время доступа к элементу массива при случайном обходе больше, чем при прямом или обратном; объяснить причины этой разницы во временах.

# ОПИСАНИЕ РАБОТЫ

Алгоритмы заполнения массива

1. Прямой обход. В каждую ячейку, начиная с нулевой, кладём номер следующеё ячейки. В последнюю записываем 0.
2. Обратный обход. В каждую ячейку, начиная с последней, записываем номер предыдущей ячейки. В нулевую записываем номер последней ячейки.
3. Случайный обход. Сначала заполняем массив числами -1, чтобы понимать, какие ячейки уже заполнены, а какие нет. Далее в нулевую ячейку записываем номер случайной ячейки, переходим в эту ячейку. Далее записываем в текущую ячейку номер случайной ячейки, в которой мы ещё не были (если генератор случайных чисел выдал номер ячейки, которую мы уже посещали, то двигаемся в ближайшую ячейку с числом -1). Переходим в новую ячейки. И т.д. заполняем до того момента, пока не останется одна ячейка с числом -1. В неё записываем 0, чтобы номера зациклились.

Графики зависимости среднего количества тиков процессора для доступа к элементам массива от размера массива получился таким:



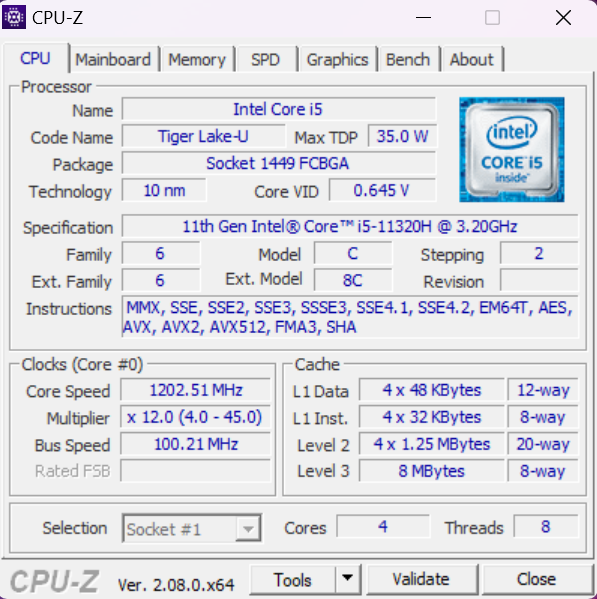
Здесь по вертикале – количество тиков процессора. По горизонтали – размер массива в Кило Байтах.

Наблюдения по графикам:

Графики прямого и обратного обходов ведут себя как график константы (y = 10)

График случайного обхода с 30КБ до 50КБ почему-то колеблется (этого быть не должно). С 50КБ да 1200КБ график постепенно поднимается до 20 тактов в среднем (это L2, а до этого был L1). Далее с 1200КБ до 6000КБ график поднимается промерно до 50 тактов (это L3). Дальше идёт переход в RAM

Характеристики кэшей системы:



# ЗАКЛЮЧЕНИЕ

При прямом и обратном обходах массива количество тактов всегда фиксированное, потому что устройство управления кэшем с помощью предвыборки хорошо определяет, что подгрузать в кэш L1.

По графику случайного обхода массива можно приблизительно определить размеры кашей L1, L2, L3.

# Приложение 1

#include <chrono>

#include <fstream>

#include <intrin.h>

#include <iostream>

#include <random>

#include <set>

#include <vector>

using namespace std;

constexpr int N\_min = 1 \* 1024 / 4; // 1 KB

constexpr int N\_max = 64 \* 1024 \* 1024 / 4; // 32 MB

const int cycles\_count = 7; // Количество повторов обхода массива

const double K = 1.1; // Коэффициент умножения размера массива

const int extra\_cycles = 3; // Дополнительные повторы цикла, чтобы избежать возможного влияния других процессов

int loadCPU()

{

    const int n = 1000;

    vector<vector<int>> A(n, vector<int>(n, 10));

    vector<vector<int>> B(n, vector<int>(n, 10));

    vector<vector<int>> C(n, vector<int>(n, 10));

    for (int i = 0; i < n; ++i)

        for (int j = 0; j < n; ++j)

            for (int k = 0; k < n; ++k)

                C[i][k] += A[i][j] \* B[j][k];

    return C[0][0];

}

void fill\_in\_1(int arr[], int arr\_size)

{

    for (int i = 0; i < arr\_size - 1; ++i)

        arr[i] = i + 1;

    arr[arr\_size - 1] = 0;

}

void fill\_in\_2(int arr[], int arr\_size)

{

    arr[0] = int(arr\_size) - 1;

    for (int i = int(arr\_size) - 1; i > 0; --i)

        arr[i] = i - 1;

}

int mod(int x, int base)

{

    if (x > 0)

        return x % base;

    return x % base + base;

}

void fill\_in\_3(int arr[], int arr\_size)

{

    for (int i = 0; i < arr\_size; ++i)

        arr[i] = -1;

    mt19937 gen(static\_cast<unsigned int>(chrono::system\_clock::now().time\_since\_epoch().count()));

    uniform\_int\_distribution<> dist(1, arr\_size - 1);

    int cur\_pos = 0;

    for (int q = 0; q < arr\_size - 1; ++q)

    {

        int next = dist(gen);

        if (arr[next] != -1 || next == cur\_pos)

        {

            int next\_r = next;

            while (arr[next\_r] != -1 || next\_r == cur\_pos)

            {

                next\_r++;

                if (next\_r == arr\_size)

                    next\_r = 0;

            }

            int next\_l = next;

            while (arr[next\_l] != -1 || next\_l == cur\_pos)

            {

                next\_l--;

                if (next\_l == 0)

                    next\_l = arr\_size - 1;

            }

            if (mod(next\_r - next, arr\_size) < mod(next\_l - next, arr\_size))

                next = next\_r;

            else

                next = next\_l;

        }

        arr[cur\_pos] = next;

        cur\_pos = next;

    }

    arr[cur\_pos] = 0;

}

int main()

{

    ofstream out("results.txt"); // Для вывода результатов

    int \*arr = new int[N\_max];

    // Разгон процессора

    if (loadCPU() == 0)

        cout << "QQ";

    for (int arr\_size = N\_min; arr\_size < N\_max; arr\_size = int(arr\_size \* K))

    {

        cout << arr\_size << "\t";

        // fill\_in\_1(arr, arr\_size); // прямой обход

        // fill\_in\_2(arr, arr\_size); // обратный обход

        fill\_in\_3(arr, arr\_size); // случайный обход

        // Прогрев кэша

        int x = 0;

        for (int i = 0; i < arr\_size \* cycles\_count; ++i)

            x = arr[x];

        if (x != 0)

            cout << "How did it happen?\n";

        unsigned long long min\_ticks = 0;

        for (int q = 0; q < extra\_cycles; ++q)

        {

            x = 0;

            unsigned long long tick\_start = \_\_rdtsc();

            for (int i = 0; i < arr\_size \* cycles\_count; ++i)

                x = arr[x];

            unsigned long long tick\_finish = \_\_rdtsc();

            if (x != 0)

                cout << "How did it happen?\n";

            unsigned long long dt = tick\_finish - tick\_start;

            if (min\_ticks == 0 || dt < min\_ticks)

                min\_ticks = dt;

        }

        out << arr\_size \* 4 / 1024 << "\t";

        out << min\_ticks / (arr\_size \* cycles\_count) << "\n";

    }

    delete (arr);

    cout << "FINISHED\n";

    return 0;

}

# 

# Приложение 2

cmake\_minimum\_required(VERSION 3.10)

project(CASH)

add\_executable(CashTest src/main.cpp)

target\_compile\_features(CashTest PUBLIC cxx\_std\_20)

add\_compile\_options(-O1)