###### 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

Степень ассоциативности кэш-памяти

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)

# ЦЕЛЬ

Экспериментальное определение степени ассоциативности кэш-памяти.

# ЗАДАНИЕ

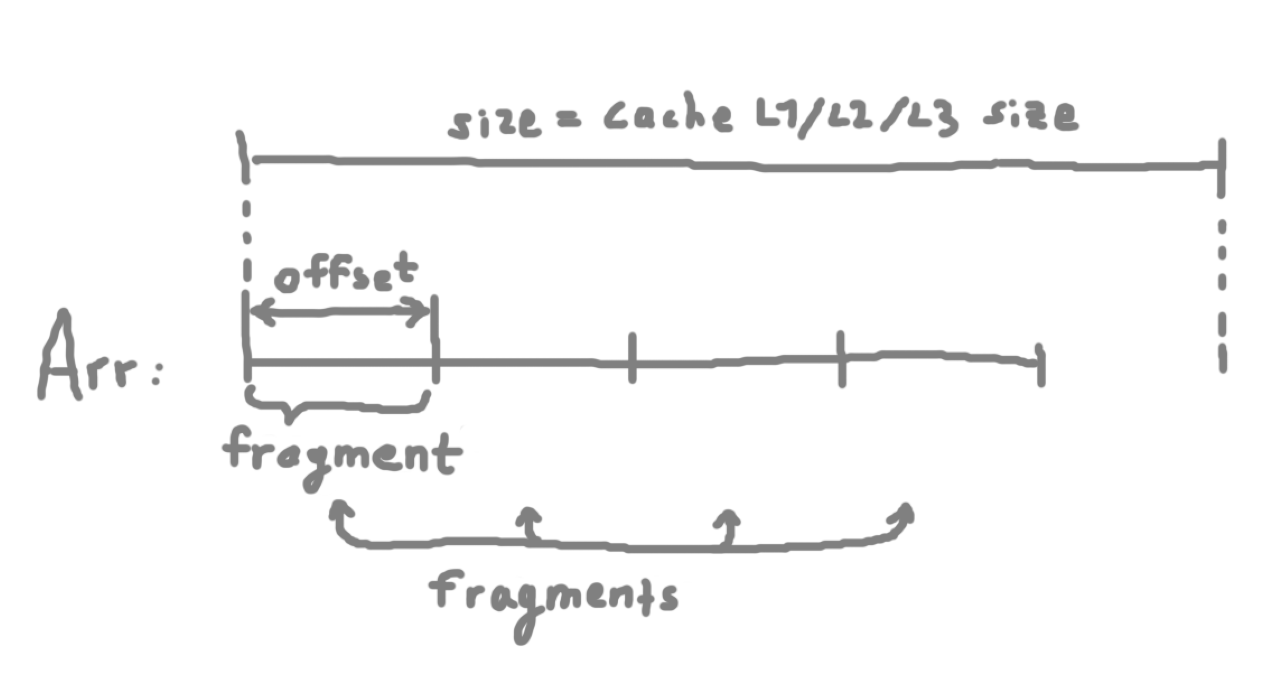
1. Написать программу, выполняющую обход памяти в соответствии с заданием.

2. Измерить среднее время доступа к одному элементу массива (в тактах процессора) для разного числа фрагментов: от 1 до 32. Построить график зависимости времени от числа фрагментов.

3. По полученному графику определить степень ассоциативности кэшпамяти, сравнить с реальными характеристиками исследуемого процессора.

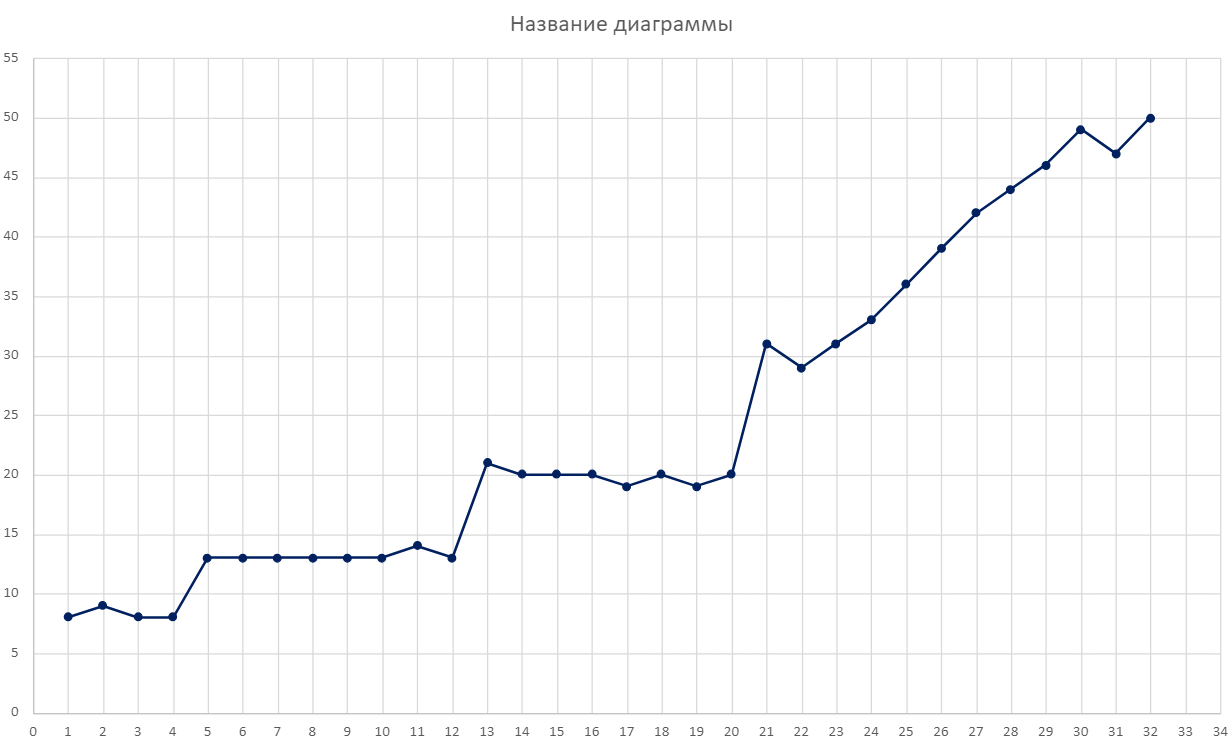
# ОПИСАНИЕ РАБОТЫ

Было принято решение сделать разбиение на фрагменты как на рисунке:

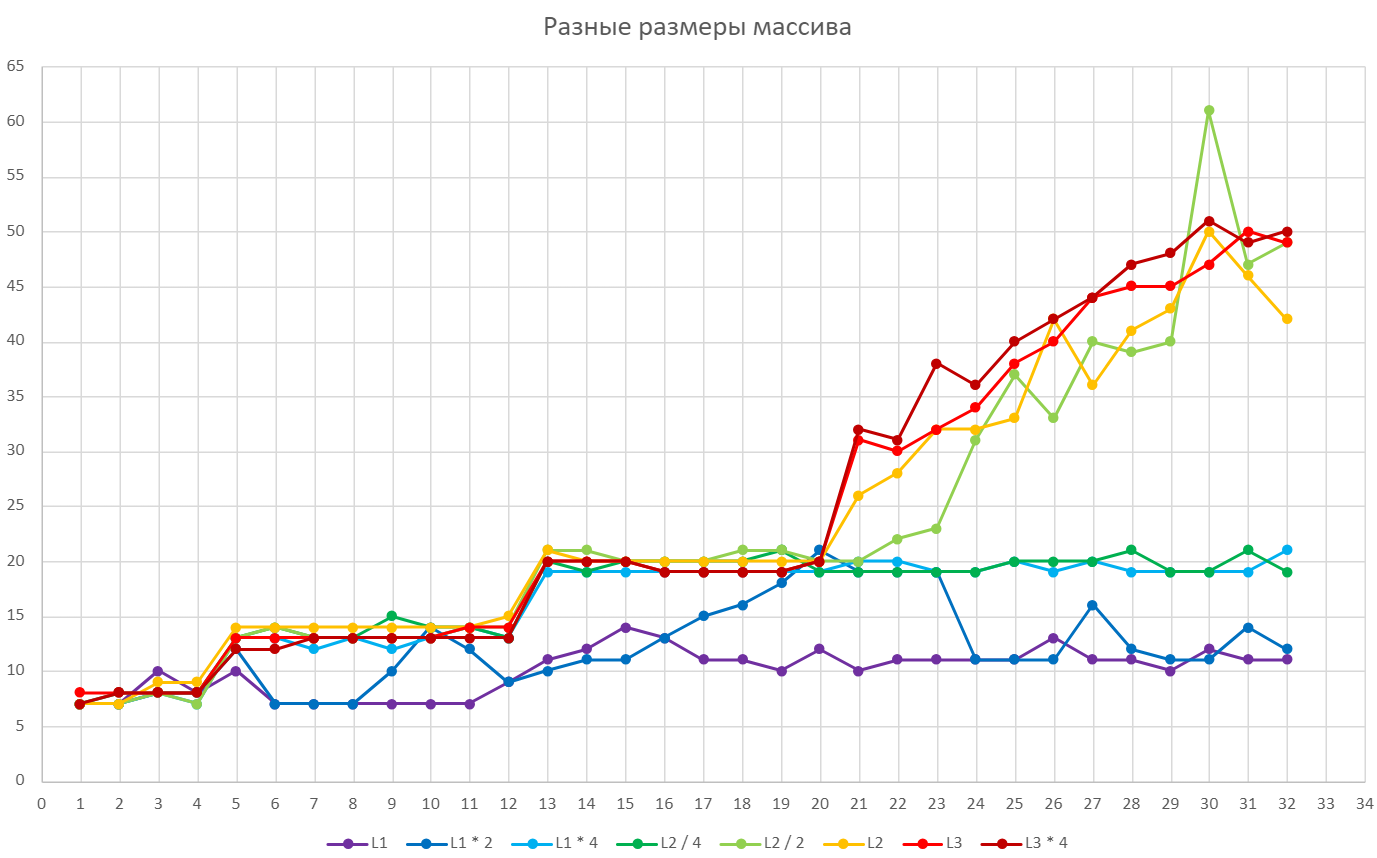


Здесь offset это максимальное число, которое меньше чем size / fragments, при этом кратное степени двойки. С таким offset-ом достигается чёткое видение пробуксовки на разных уровнях кэша (пробуксовка будет, если fragments больше чем степень ассоциативности определённого уровня кэша). Каждый фрагмент полностью заполняет соответствующий участок длины offset (чем больше фрагмент, тем на большем количестве множеств будет пробуксовка. Это даёт большую точность). Размер массива это fragments \* offset.

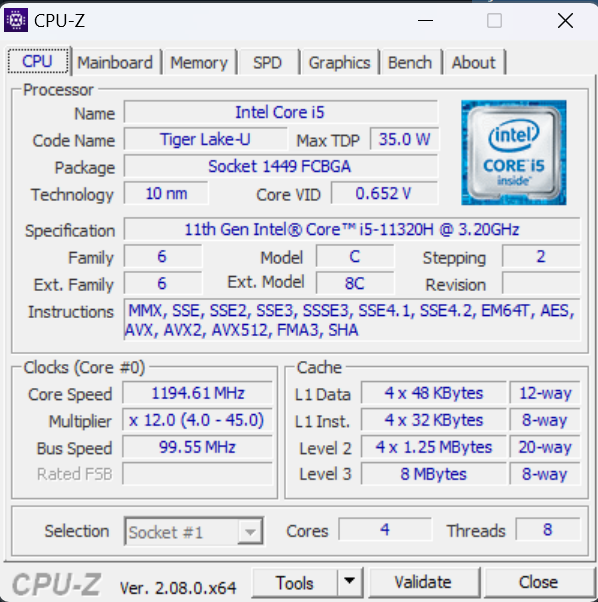
Для размера массива чуть меньше чем размер L3 получился следующий график:



Потом было проведено ещё несколько тестов для разных размеров массива:



Реальные степени ассоциативности можно посмотреть здесь:



Судя по графикам, я бы сказал, что 4 – это степень ассоциативности буфера трансляции адресов, 12 – степень ассоциативности L1 и L2, 20 - степень ассоциативности L3. Но в реальности оказалось, что степень ассоциативности L1 это 12, L2 – 20, L3 – 8.

# ЗАКЛЮЧЕНИЕ

У разных процессоров разная архитектура, и нельзя, судя по графикам, сто процентно утверждать, какой уровень ассоциативности у каждого уровня кэша.

Чем больше размер массива, тем выше график. Причём при малом размере буфера кэш ведёт себя как полностью ассоциативный, поэтому график имеет не ступенчатый вид (синий и фиолетовый графики на рисунке выше)

# Приложение 1

main.cpp:

#include <Windows.h>

#include <fstream>

#include <iostream>

#include <new>

#include <vector>

constexpr int CacheL1Size = 48 \* 1024;       // L1 - 48KB

constexpr int CacheL2Size = 1280 \* 1024;     // L2 - 1.25MB

constexpr int CacheL3Size = 8 \* 1024 \* 1024; // L3 - 8MB

const int Loops = 5;                         // Для большей точности

using namespace std;

int loadCPU()

{

    const int n = 500;

    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 fiilInArr(int \*arr, const int fragments, const int fragmentSize, const int offset)

{

    for (int i = 0; i < fragmentSize; ++i)

    {

        for (int j = 0; j < fragments - 1; ++j)

        {

            arr[j \* offset + i] = (j + 1) \* offset + i;

        }

        arr[(fragments - 1) \* offset + i] = i + 1;

    }

    arr[(fragments - 1) \* offset + fragmentSize - 1] = 0;

}

int calculateOffset(int size, int fragments)

{

    int limit = size / fragments;

    int offset = 1;

    while (offset < limit)

        offset \*= 2;

    return offset / 2;

}

int main()

{

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

    loadCPU();

    for (int fragments = 1; fragments <= 32; ++fragments)

    {

        cout << fragments << " ";

        const int size = CacheL3Size;

        int \*arr = new int[size];

        int offset = calculateOffset(size, fragments);

        int fragmentSize = offset;

        fiilInArr(arr, fragments, fragmentSize, offset);

        int pos = 0;

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

        for (int i = 0; i < fragmentSize \* fragments \* Loops; ++i)

            pos = arr[pos];

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

        if (pos != 0)

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

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

        out << dt / (fragmentSize \* fragments \* Loops) << "\n";

        delete[] arr;

    }

    return 0;

}

CMakeLists.txt:

cmake\_minimum\_required(VERSION 3.10)

project(associativity)

add\_executable(${PROJECT\_NAME} main.cpp)