

**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 ЛАБОРАТОРНОЙ РАБОТЫ**

**«Измерение степени ассоциативности кэш-памяти»**

**Студента 2 курса, 21211 группы**

**Петрова Сергея Евгеньевича**

**Направление 09.03.01 – «Информатика и вычислительная техника»**

**Преподаватель:  
Антон Юрьевич Кудинов**

**Новосибирск 2022**

## СОДЕРЖАНИЕ

<i>СОДЕРЖАНИЕ</i>	2
<i>ЦЕЛЬ</i>	3
<i>ЗАДАНИЕ</i>	3
<i>ОПИСАНИЕ РАБОТЫ</i>	4
<i>Пошаговое описание выполненной работы</i>	4
<i>Команды для компиляции</i>	4
<i>Результаты измерения</i>	4
<i>ЗАКЛЮЧЕНИЕ</i>	5
<i>ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)</i>	6
<i>main.cpp</i>	6
<i>CMakeLists.txt</i>	7

## ЦЕЛЬ

- Экспериментальное определение степени ассоциативности кэш-памяти;

## ЗАДАНИЕ

- Написать программу, выполняющую обход данных в памяти, который вызывает «буксование» кэш-памяти;
- 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;
- По полученному графику определить степень ассоциативности кэш-памяти, сравнить с реальными характеристиками исследуемого процессора;
- Составить отчет по лабораторной работе. Отчет должен содержать следующее:
  - Титульный лист;
  - Цель лабораторной работы;
  - Параметры теста: размер фрагментов, величина смещения.
  - График зависимости среднего времени доступа к элементу массива от числа фрагментов;
  - Реальные и полученные в ходе тестирования значения степени ассоциативности кэш-памяти процессора;
  - Полный компилируемый листинг реализованной программы и команды для ее компиляции;
  - Вывод по результатам лабораторной работы.

## ОПИСАНИЕ РАБОТЫ

### Пошаговое описание выполненной работы

1. Написал программу, выполняющую обход памяти в соответствии с заданием;
2. Измерил среднее время доступа к одному элементу массива (в тактах процессора) для числа фрагментов от 1 до 32 при следующих параметрах:
  - размер фрагментов - 12 МБ;
  - смещение между фрагментами - 24 МБ;
3. Построил график зависимости времени чтения элемента массива от числа фрагментов (см. [Результаты измерения](#));
4. Нашёл информацию о кэш-памяти процессора [Intel Xeon X5660](#);

Level 1 cache size ?	6 x 32 KB 4-way set associative instruction caches 6 x 32 KB 8-way set associative data caches
Level 2 cache size ?	6 x 256 KB 8-way set associative caches
Level 3 cache size	12 MB 16-way set associative shared cache

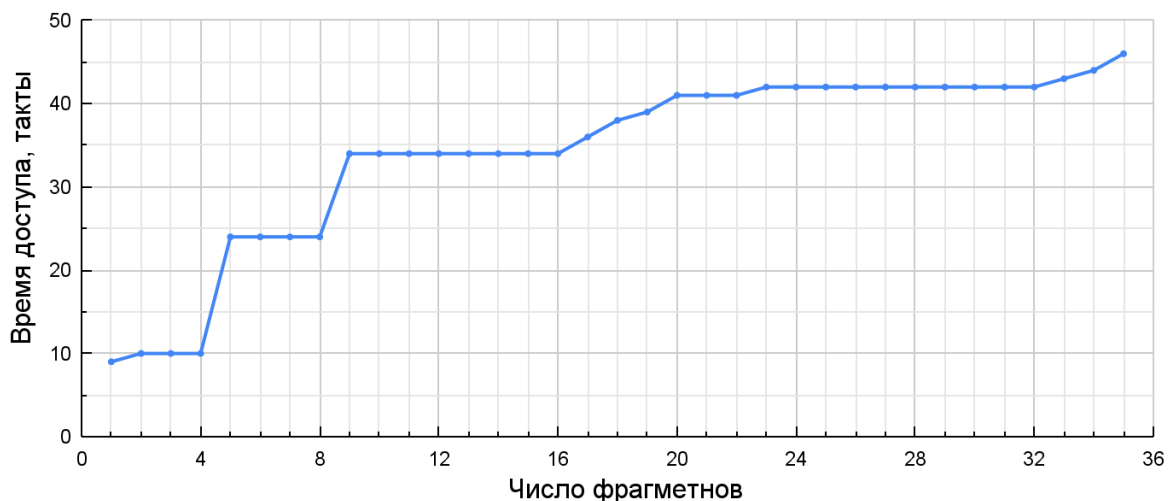
### Команды для компиляции

```
evmpu@comrade:~$ /usr/bin/cmake
-DCMAKE_BUILD_TYPE=Debug
-G "CodeBlocks - Unix Makefiles"
-S /home/evmpu/21211/s.petrov1/lab9
-B /home/evmpu/21211/s.petrov1/lab9/cmake-build-debug

evmpu@comrade:~$ /usr/bin/cmake
--build /home/evmpu/21211/s.petrov1/lab9/cmake-build-debug
--target lab9
```

### Результаты измерения

Зависимость времени чтения элемента массива от  
числа фрагментов



## **ЗАКЛЮЧЕНИЕ**

*В ходе выполнения лабораторной работы:*

- Экспериментально определялись степени ассоциативности кэш-памяти;

*По результатам проведённых исследований можно сделать следующие выводы:*

- По приросту времени доступа к данным в памяти можно определить степени ассоциативности кэш-памяти разных уровней (L1 - 8-way, L2 - 8-way, L3 - 16-way);
- Прирост времени доступа к данным в памяти после 4 фрагментов связан с тем, что степень ассоциативности буфера преобразования виртуальных адресов в физические (TLB) равен 4.

## ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)

*main.cpp*

```
#include <cstdint>
#include <iomanip>
#include <iostream>

#define SIZE      (32 * 24 * 1024 * 1024 / sizeof(uint32_t))
#define CACHE_SIZE (12 * 1024 * 1024 / sizeof(uint32_t))
#define OFFSET    (24 * 1024 * 1024 / sizeof(uint32_t))

union Time
{
    uint64_t m64;
    struct { uint32_t l, h; } m32;
};

void Clock(Time & time);
uint64_t Bypass(const uint32_t * array, size_t fragments,
                size_t cache_size);
void InitArray(uint32_t * array, size_t fragments, size_t offset,
                size_t cache_size);

int main()
{
    auto * array = new uint32_t[SIZE];

    std::cout << std::left << std::setw(20) << "Fragments: "
               << "Tacts: " << std::endl;

    InitArray(array, 32, OFFSET, CACHE_SIZE);
    Bypass(array, 32, CACHE_SIZE);

    for(int fragments = 1; fragments <= 32; ++fragments)
    {
        InitArray(array, fragments, OFFSET, CACHE_SIZE);
        std::cout << std::left << std::setw(20) << fragments
                  << Bypass(array, fragments, CACHE_SIZE)
                  << std::endl;
    }

    delete [] array;
    return EXIT_SUCCESS;
}

void Clock(Time & time)
{
    asm( "rdtsc\n" : "=a" (time.m32.l), "=d" (time.m32.h) );
}

void InitArray(uint32_t * array, size_t fragments, size_t offset,
                size_t cache_size)
{
    for(size_t i = 0; i < cache_size; ++i)
    {
```

```

        for(size_t j = 0; j < fragments; ++j)
            array[i + j * offset] = i + (j + 1) * offset;

        array[i + (fragments - 1) * offset] = i + 1;
    }

    array[(cache_size - 1) + (fragments - 1) * offset] = 0;
}

uint64_t Bypass(const uint32_t * array, size_t fragments,
                size_t cache_size)
{
    uint64_t t_min = UINT64_MAX;
    Time start{}, end{};

    for (int i = 0; i < 10; ++i)
    {
        Clock(start);
        for (uint32_t k = 0, j = 0; j < fragments * cache_size; ++j)
            k = array[k];
        Clock(end);

        if (t_min > end.m64 - start.m64)
            t_min = end.m64 - start.m64;
    }

    return t_min / (fragments * cache_size);
}

```

### *CMakeLists.txt*

```

cmake_minimum_required(VERSION 3.16.3)
project(lab9 CXX)
add_executable(lab9 main.cpp)

```