

**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
ИЗМЕРЕНИЕ СТЕПЕНИ АССОЦИАТИВНОСТИ КЭШ-ПАМЯТИ

Студентки 2 курса, группы 21205

Евдокимовой Дарьи Евгеньевны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук, доцент
А.Ю. Власенко

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ.....	3
ЗАДАНИЕ.....	3
ОПИСАНИЕ РАБОТЫ.....	4
СПИСОК ЛИТЕРАТУРЫ.....	7
ЗАКЛЮЧЕНИЕ.....	8
Приложение 1. Листинг программы определения ассоциативности кэша....	9
Приложение 2. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L1.....	11
Приложение 3. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L2.....	12
Приложение 4. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L3.....	13

ЦЕЛЬ

- 1) Экспериментальное определение степени ассоциативности кэш-памяти.

ЗАДАНИЕ

- 1) Написать программу, выполняющую обход памяти в соответствии с заданием к лабораторной работе.
- 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 кэш-памяти, сравнить с реальными характеристиками исследуемого процессора.

ОПИСАНИЕ РАБОТЫ

В ходе задания использовался компьютер с архитектурой amd64, с операционной системой Ubuntu 20.04.5 LTS и процессором Intel® Core™ i3-6100U CPU @ 2.30GHz × 4.

Команда для определения размера кэш-строки (и в целом информация про процессор):

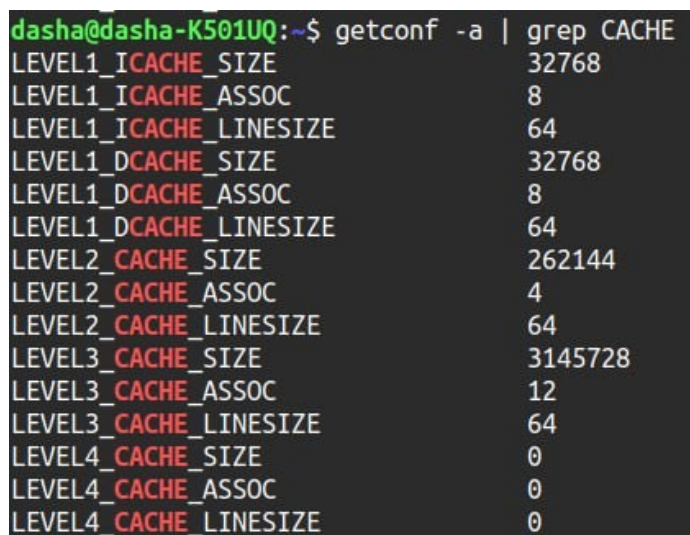
```
SYSNODE=/sys/devices/system/node
```

```
grep '.*' $SYSNODE/node*/cpu*/cache/index/* 2>/dev/null | awk '-F[:/]' '{ printf "%6s %6s %24s %s\n" $6, $7, $9, $10, $11 ; }'
```

Последовательное описание работы

1) Узнаем размеры разных уровней кэш-памяти (см. Рис. 1) с помощью команды конфигурации

```
getconf -a | grep CACHE
```



```
dasha@dasha-K501UQ:~$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE          32768
LEVEL1_ICACHE_ASSOC         8
LEVEL1_ICACHE_LINESIZE      64
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
LEVEL2_CACHE_SIZE           262144
LEVEL2_CACHE_ASSOC          4
LEVEL2_CACHE_LINESIZE       64
LEVEL3_CACHE_SIZE           3145728
LEVEL3_CACHE_ASSOC          12
LEVEL3_CACHE_LINESIZE       64
LEVEL4_CACHE_SIZE            0
LEVEL4_CACHE_ASSOC           0
LEVEL4_CACHE_LINESIZE       0
```

Рис. 1 Определение размера кэш-памяти

2) По выведенным данным получаем следующую таблицу (Табл. 1)

Табл. 1

Уровень кэш-памяти	Размер кэша, в байтах
L1	32768
L2	262144
L3	3145728

3) В современных процессорах, в т.ч. на процессоре Intel i3-6100U используется множественно-ассоциативный кэш¹.

- 4) Для того, чтобы определить степень ассоциативности кэш-памяти нужно сделать такой обход данных, который вызовет её «буксование». Для этого необходимо организовать доступ в память по адресам, отображаемым на одно и то же множество в кэш-памяти. Увеличим количество таких конфликтующих обращений. Когда оно превысит степень ассоциативности кэш-памяти, то данные начнут вытесняться из кэш-памяти, и возникнет кэш-буксование, которое приводит к увеличению времени выполнения программы, благодаря чему можно определить степень ассоциативности кэш-памяти.

Описание идеи эксперимента

Заведём большой массив, в котором данными будут заполнены «блоки» (fragments), отстоящие друг от друга на одно и то же расстояние. Элементы массива образуют связный список. Обход будет устроен следующим образом: сначала обращаемся ко всем первым элементам в каждом фрагменте, затем – ко всем вторым, потом – третьим и так далее. Размер фрагмента может принимать значение от 1 до длины кэш-строки. По заданию число фрагментов изменяется от 1 до 32.

Для того, чтобы адреса отображались на одно и то же множество в кэш-памяти, необходимо, чтобы смещение между ними было кратно размеру банка кэш-памяти, но его размер не известен. Зато мы знаем, размер кэш-памяти кратен размеру банка кэш-памяти.

То есть если мы подберем такое значение смещения, которое является наименьшим общим кратным размера банков, то сможем отследить сразу все уровни кэш-памяти.

Описание практической части

Перед запуском программы прогреем кэш программой из предыдущей лабораторной работы для того, чтобы процессор с динамически изменяемой частотой установил её на фиксированном уровне.

Программа для запуска: *cacheAssoc.cpp*

Команда компиляции: *g++ -o cache cacheAssoc.cpp -O1*

Команда запуска: *./cache*

Обход по массиву будем делать 100 раз для более точного определения времени обращения к элементу.

Для каждого уровня кэш-памяти будем проводить замеры времени в зависимости от размеров этой кэш-памяти.

Начнём с первого кэша L1, размер которого равен 32768 байт (32Кбайт). И пусть смещение равно 32768 байт. Полученные результаты представлены в виде графика в Приложении 2.

По графику однозначно видно, что степень ассоциативности первого уровня кэш-памяти равна 8, т.к. «скачок» наблюдается после восьми тактов.

Разберемся со вторым кэшем L2, размер которого равен 262144 байта (256 Кбайт). Этому же значению пусть равно и смещение.

Полученные результаты представлены в виде графика в Приложении 3. По графику видно, что степень ассоциативности второго уровня кэш-памяти равна 4, т.к. «скачок» наблюдается после четырёх тактов, а скачок после 8 тактов — это L1.

Разберемся с третьим уровнем кэша. Экспериментально узнать размер L3 не получилось (см. Приложение 4). На графике от 1го до 31го фрагмента постоянная на 5 тактов, затем скачок до 6 тактов, хотя реальная степень ассоциативности L3 на процессоре Intel i3-6100U равна 12.

12 - это не степень двойки, возможно, проблема определения размера кэша в этом. Потому что чаще всего степень ассоциативности суть степень двойки. Не было скачка после 12 фрагмента.

Можем предположить, что размер L3 нельзя определить на данной машине, потому что кэш L3 распределен между всеми ядрами². И нет способа как-то «выделить» ядро только для уровня L3.

СПИСОК ЛИТЕРАТУРЫ

1. Статья про кэш и ассоциативность в процессорах Intel:
URL: <https://eprint.iacr.org/2015/905.pdf> .
2. Статья про распределение кэша L3 между ядрами.
URL: <https://cpuninja.com/cpu-cache/> .

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы я узнала разные команды на Linux для определения информации о процессоре. Так же узнала, что такое «эффект буксования», ассоциативность кэш-памяти и как её определять.

По итогу работы определила уровни кэш-памяти своего процессора и сравнила результаты с реальными значениями. А еще узнала, что ассоциативность кэша первого уровня может быть больше ассоциативности второго уровня.

Приложение 1. Листинг программы определения ассоциативности

кэша

```
#include <climits>
#include <iostream>
#include <x86intrin.h>

void GoThroughArray(size_t *array, size_t offset,
                    size_t amountFrag) {
    for (size_t idx = 0; idx < offset; idx++) {
        for (size_t frag = 0; frag < amountFrag - 1; frag++) {
            array[frag * offset + idx] = (frag + 1) * offset + idx;
        }
        array[(amountFrag - 1) * offset + idx] = (idx + 1) % offset;
    }
}

size_t CountTimeInTics(size_t *array, size_t offset) {
    const size_t amountBypass = 100;
    size_t elem = 0;

    size_t start = __rdtsc();
    for (size_t bypass = 0; bypass < amountBypass * offset; bypass++) {
        elem = array[elem];
    }
    size_t end = __rdtsc();
    if (elem == -1) {
        std::cout << "WoW! I am out of bounds!";
    }

    size_t tmpTime = end - start;
    size_t minTime = INT32_MAX;

    minTime = minTime > tmpTime ? tmpTime : minTime;

    return minTime / (amountBypass * offset);
}

int main() {
    const size_t maxNumOfFrag = 32;

    const int L1_CACHE_SIZE = 32768;
    const int coeffL1 = 8; //offset should be divisible by bank size
                           //coeff should be pow of 2
    const size_t offsetL1 = L1_CACHE_SIZE / coeffL1;
    size_t *arrayL1 = new size_t[maxNumOfFrag * offsetL1]();
    for (size_t fragm = 1; fragm <= maxNumOfFrag; fragm++) {
        std::cout << "L1 ";
        std::cout << "fragment " << fragm << " ";
    }
```

```

        GoThroughArray(arrayL1, offsetL1, fragm);
        std::cout << CountTimeInTics(arrayL1, offsetL1 * fragm) << " tics"
                << std::endl;
    }
    delete [] arrayL1;

    std::cout << "=====" <<std::endl << std::endl;

    const size_t L2_BANK_SIZE = 262144;
    const int coeffL2 = 16;
    const size_t offsetL2 = L2_BANK_SIZE / coeffL2;
    size_t *arrayL2 = new size_t[maxNumOfFragms * offsetL2]();
    for (size_t fragm = 1; fragm <= maxNumOfFragms; fragm++) {
        std::cout << "L2 ";
        std::cout << "fragment " << fragm << " ";

        GoThroughArray(arrayL2, offsetL2, fragm);
        std::cout << CountTimeInTics(arrayL2, offsetL2 * fragm) << " tics"
                << std::endl;
    }
    delete [] arrayL2;

    std::cout << "=====" <<std::endl << std::endl;

    //CODE BELOW DOESN'T COUNT L3 ASSOCIATIVITY
    /*
    const size_t L3_BANK_SIZE = 3145728;
    const size_t offsetL3 = L3_BANK_SIZE / 1024 / 1024 / 8;
    size_t *arrayL3 = new size_t[maxNumOfFragms * L3_BANK_SIZE]();
    for (size_t fragm = 1; fragm <= maxNumOfFragms; fragm++) {
        std::cout << "L3 ";
        std::cout << "fragment " << fragm << " ";
        GoThroughArray(arrayL3, offsetL3, fragm);

        std::cout << CountTimeInTics(arrayL3, offsetL3 * fragm) << " tics"
                << std::endl;
    }
    delete [] arrayL3;
    */
}

```

Приложение 2. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L1

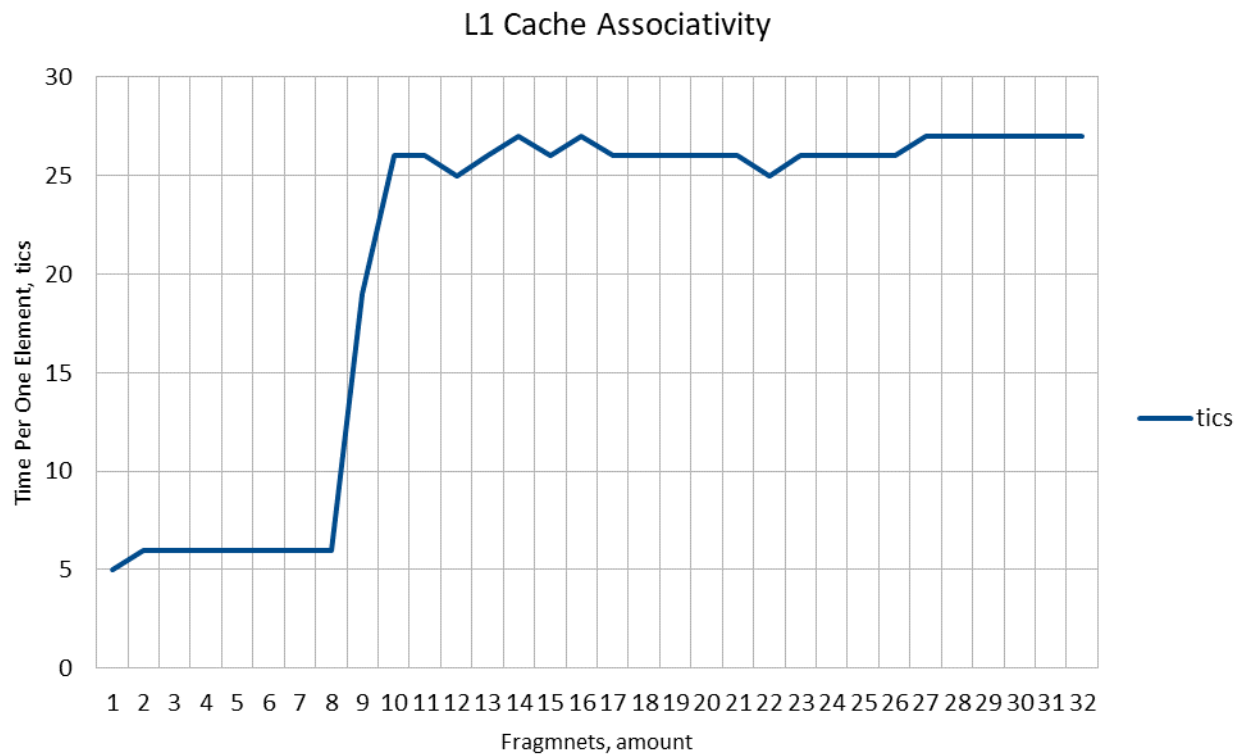


Рис. 1. График для L1, offset = 4096 Кб

Приложение 3. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L2

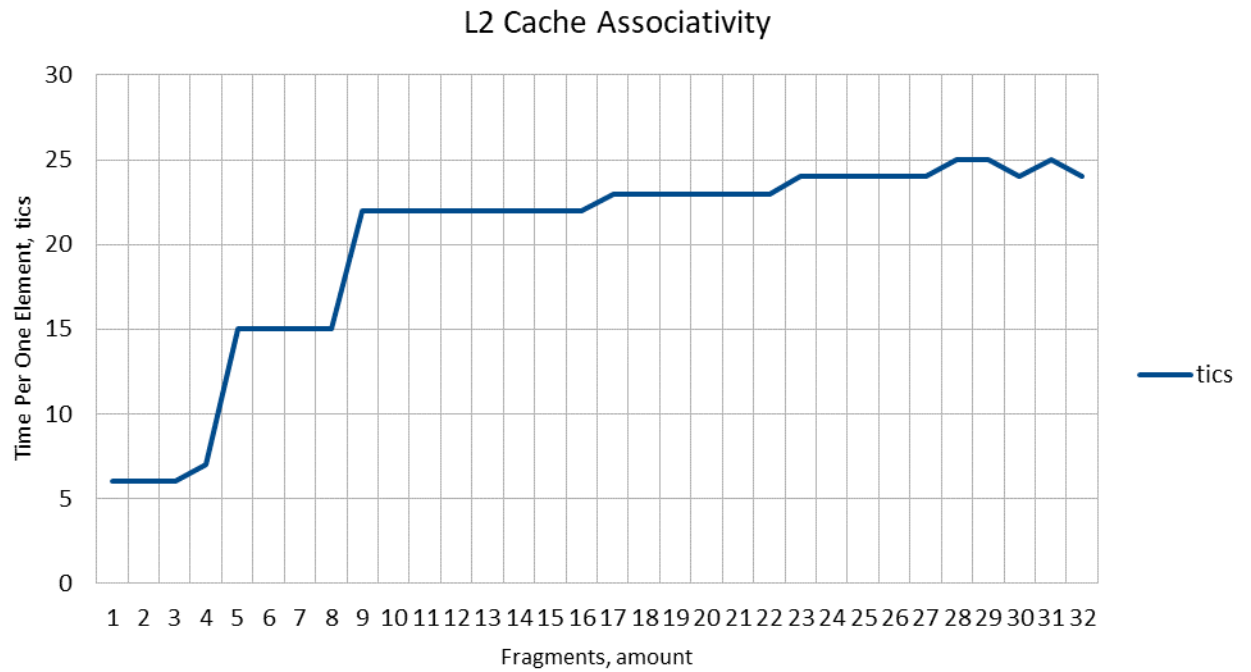


Рис. 1. График для L2, offset = 256 Кб

Приложение 4. График зависимости времени обращения к одному элементу от количества фрагментов для кэша L3

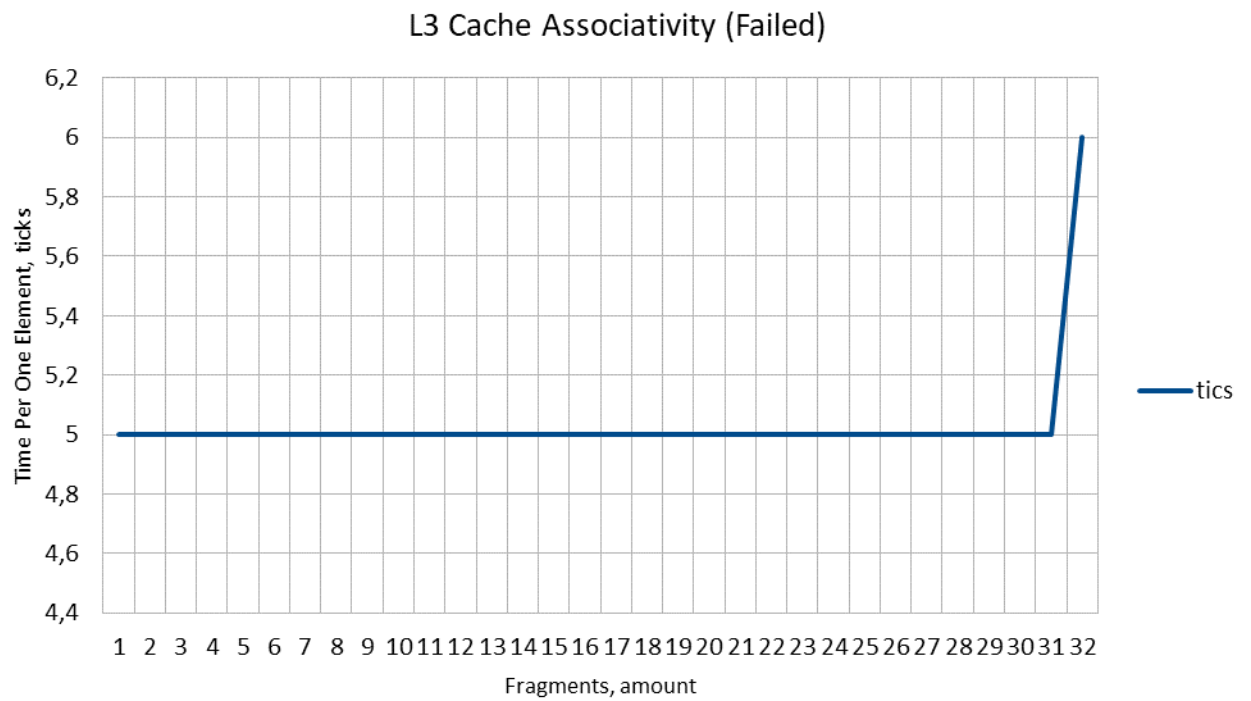


Рис. 1. График для L3, offset = 3 Мб