**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**

«Ассоциативность кэша процессора»

студента 2 курса, группы 20205

**Муратова Максима Александровича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Доцент
Власенко А. Ю.

Новосибирск 2021

# СОДЕРЖАНИЕ

## ЦЕЛИ

- Узнать степень ассоциативности своего кэша;
- Определить степень ассоциативности кэша экспериментально.

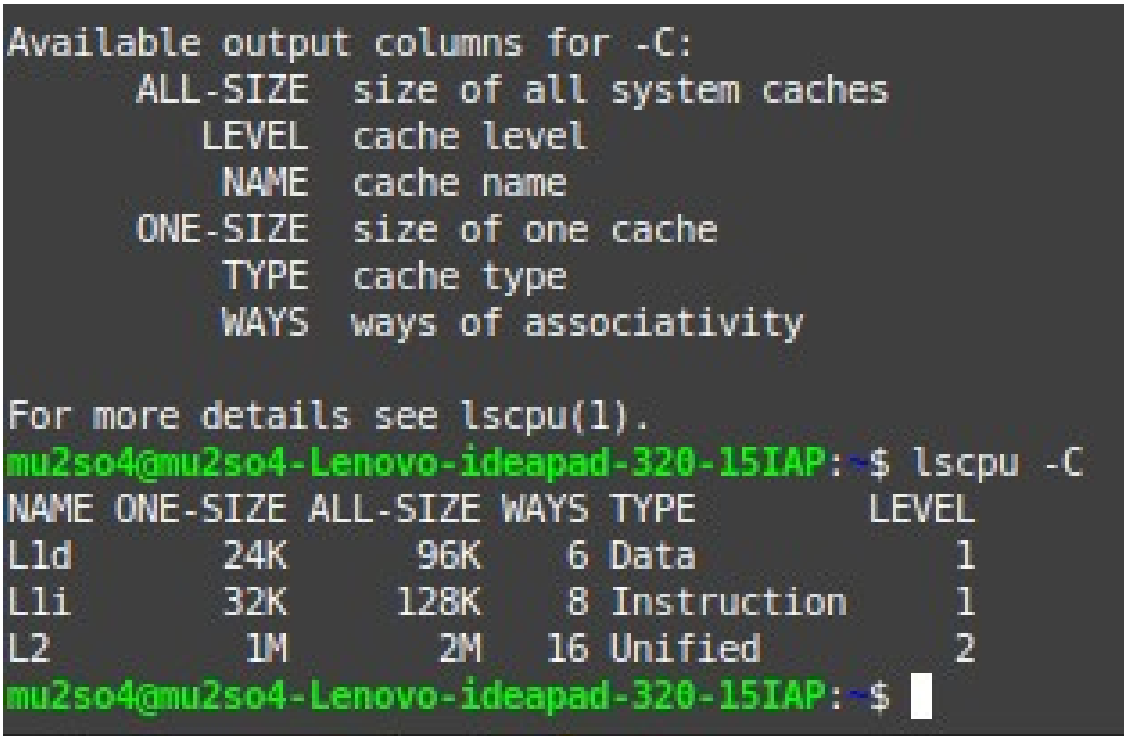
## ЗАДАНИЕ

1. При помощи утилиты найти реальные вначения ассоциативности кэша;
2. С помощью эффекта пробуксовки кэша экспериментально найти степень ассоциативности кэша;
3. Сравнить экспериментальные данные с реальными.

# ОПИСАНИЕ РАБОТЫ

Для начала найдём степень ассоциативноти кэша из документации. Для этого используем команду:

```
$ lscpu -C
```




Степень ассоциативности указана в столбце WAYS. Таким образом, степени ассоциативности L1d = 6, L2 = 16.

Как можно заметить, на устройстве, на базе которого был проведён эксперимент, нет кэша L3.

Теперь напишем саму программу для разных обхода массива

Для удобства составления программы, она была поделена на 5 частей:

1. `main.cpp` – собственно реализация алгоритма обхода массива (ПРИЛОЖЕНИЕ 1);
2. `mtrx.h`, `mtrx.cpp` – класс `Matrix`, с помощью которого разгонялся процессор (ПРИЛОЖЕНИЯ 2 и 3 соответственно);
3. `traversal.h`, `tow.cpp` – обхода массива, который и вызовет пробуксовку кэша (ПРИЛОЖЕНИЯ 4 и 5 соответственно).

Для начала необходимо найти наименьший сдвиг, который при проходе 8 элементов вызовет пробуксовку. Для этого специально был написан скрипт `offset_test.sh` (ПРИЛОЖЕНИЕ 7). Он прогоняет при данном смещении массив 100 раз в три круга, результат выводится в таблицу.

Из трёх попыток выбирается та, что с наименьшим числом тактов. Для запуска скрипта достаточно ввести `$ ./offset_test.sh`

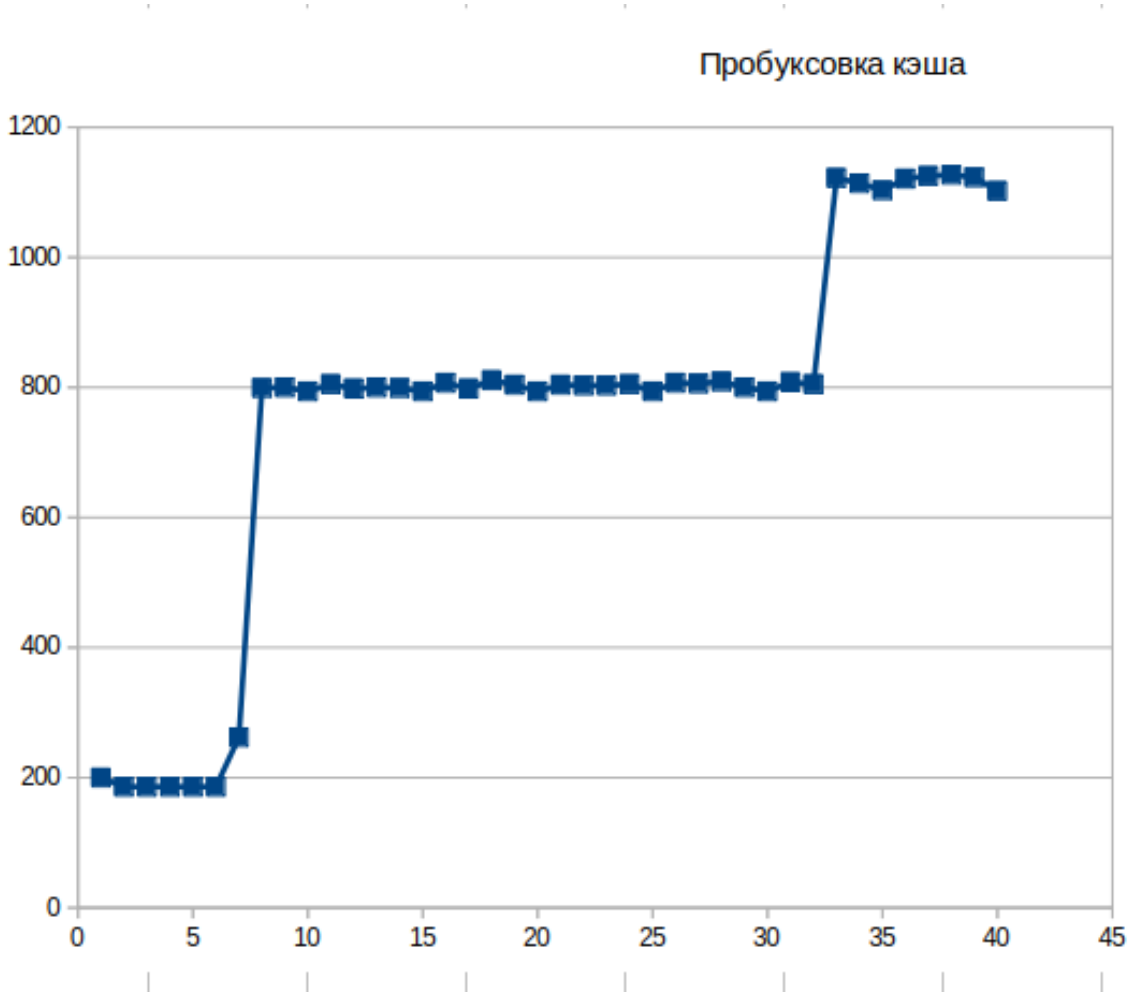
По нему определили, что требуемое смещение составляет 1024 переменных типа int, или 4 КиБ. С протоколом тестирования можно ознакомиться в ПРИЛОЖЕНИИ 9.

Это нужно было для главной цели – при фиксированном смещении прогнать массив с увеличением числа блоков, от 1 до 40. Это было выполнено при помощи скрипта `tester.sh` (ПРИЛОЖЕНИЕ 8). Алгоритм был тот же: 100 прогонов в три круга, из них берём с наилучшим результатом. Протокол этого тестирования доступен в ПРИЛОЖЕНИИ 10.

Для запуска теста достаточно ввести `$ ./test.sh 10` , где 10 – найденное смещение в элементах типа `int`.

Отдельно собирать проект не нужно: это включено в оба теста.

Таблично данные выглядят так:




По Ox – число фрагментов заданного размера (здесь это 4 КиБ), по Oy – среднее время доступа к 1 элементу массива на 100 прогонов в тактах.

Из данной таблицы можно сделать следующие микровыводы:

1. На шаге 7 произошла первая пробуксовка, таким образом, экспериментальная степень ассоциативности L1d – 6.

2. На шаге 33 произошла вторая пробуксовка, таким образом, экспериментальная степень ассоциативности L2 – 32.

Таким образом, экспериментальная степень ассоциативности L1d совпала с теоретической, а L2 вдвое больше (либо там требовалось вдвое большее смещение).

# ЗАКЛЮЧЕНИЕ

1. Экспериментальные данные степеней ассоциативности кэшей L1d и L2 совпали с теоретическими данными либо отличались от них на степень двойки;
2. Смещение, способное вызвать пробуксовку кэша – 4 Киб для L1d и 8 Киб для L2.

**Приложение 1.** *main.cpp*
*основной текст программы*

```
#include <iostream>
#include <fstream>
#include "mtrx.h"
#include "traversal.h"

int main(int argc, char **argv) {
    if(argc != 4) {
        std::cerr << "Wrong input\n";
        return 0;
    }

    union ticks {
        unsigned long long t64 = 0;
        struct s32 { long th, tl; } t32;
    } start, end;

    int size = atoi(argv[1]), offset = atoi(argv[2]),
watchingCount = atoi(argv[3]);
    int *arr = new int[size];
    createLoop(arr, size, offset);

    //acceleration
    auto *a = new Matrix{13}, *b = new Matrix{*a};
    for(int index = 0; index < 410000; index++)  //410'000 is
the perfect count
        *a *= *b;
    delete a;
    delete b;
```

```
    std::ofstream out("/dev/null");
    //heating
    int pos = 0;
    for(int index = 0; index < size * watchingCount; index++) {
        pos = arr[pos];
        if(pos == 314) out << "100 Pi\n";
    }

    asm("rdtsc\n":"=a"(start.t32.th),"=d"(start.t32.tl));
    for(int index = 0; index < size * watchingCount; index++) {
        pos = arr[pos];
        if(pos == 314) out << "100 Pi\n";
    }
    asm("rdtsc\n":"=a"(end.t32.th),"=d"(end.t32.tl));

    std::cout << size << '\t' << offset << '\t' << (end.t64 -
start.t64) / (unsigned long long) size << '\t';

    asm("rdtsc\n":"=a"(start.t32.th),"=d"(start.t32.tl));
    for(int index = 0; index < size * watchingCount; index++) {
        pos = arr[pos];
        if(pos == 314) out << "100 Pi\n";
    }
    asm("rdtsc\n":"=a"(end.t32.th),"=d"(end.t32.tl));

    std::cout << (end.t64 - start.t64) / (unsigned long long)
size << '\t';

    asm("rdtsc\n":"=a"(start.t32.th),"=d"(start.t32.tl));
    for(int index = 0; index < size * watchingCount; index++) {
        pos = arr[pos];
        if(pos == 314) out << "100 Pi\n";
    }
```

```
    asm("rdtsc\n":"=a"(end.t32.th),"=d"(end.t32.tl));


    std::cout << (end.t64 - start.t64) / (unsigned long long)
size << '\n';


    out.close();

    delete[] arr;

    return 0;

}
```

## ***Приложение 2.*** `mtrx.h`

### *заголовочный файл класса* `Matrix`

```
#ifndef VECTOR_MTRX_H
#define VECTOR_MTRX_H

#include <iostream>

class Matrix {
    int size;
    float *data;
public:
    explicit Matrix(int length);
    ~Matrix();
    Matrix(const Matrix &otherMatrix);

    Matrix &operator*=(const Matrix &b);
};
#endif //VECTOR_MTRX_H
```

***Приложение 3.*** *mtrx.cpp*

*реализация основных методов класса* *Matrix*

```
#include "mtrx.h"

Matrix::Matrix(int length): size(length) {
    data = new float[size * size];
    for(int index = 0; index < size; index++) {
        data[index * (size + 1)] = 1;
    }
}

Matrix::~Matrix() {
    delete[] data;
    data = nullptr;
}

Matrix::Matrix(const Matrix &otherMatrix):
        size(otherMatrix.size)
{
    data = new float[size * size];
    for(int index = 0; index < size * size; index++)
        data[index] = otherMatrix.data[index];
}

Matrix &Matrix::operator*=(const Matrix &b) {
    if(size == b.size) {
        auto *newData = new float[size * size];
        if(size == 4) newData[4] = 0; //else I will get a nan
        for(int x = 0; x < size; x++) {
            for(int y = 0; y < size; y++) {
                for(int i = 0; i < size; i++)
```

```
                    newData[x * size + y] += data[x * size + i]
* b.data[i * size + y];
            }
        }
        delete[] data;
        data = newData;
    }
    return *this;
}
```

## ***Приложение 4.*** `traversal.h`
## *заголовочный файл для функции обхода массива*

```
#ifndef MY_CASH_TRAVERSAL_H
#define MY_CASH_TRAVERSAL_H

void createLoop(int *arr, int size, int offset);

#endif //MY_CASH_TRAVERSAL_H
```

***Приложение 5.*** *tow.cpp*

*реализация обхода массива, способного вызвать пробуксовку кэша*

```
#include "traversal.h"

void createLoop(int *arr, int size, int offset) {
 if(offset >= size) {
      arr[size - 1] = 0;
      for(int index = 0; index < size - 1; index++)
           arr[index] = index + 1;
      return;
 }

 for(int index = 0; index < size - offset; index++)
      arr[index] = index + offset;

 for(int index = 0; index < offset; index++)
      arr[size - offset + index] = index;

}
```

## ***Приложение 6.*** *Makefile*
*Файл сборки программы*

```
CC=g++
CFLAGS=-O1 -c -Wall

all: associativity

associativity: tow.o main.o mtrx.o
	$(CC) main.o mtrx.o tow.o -o associativity

printArr: print.o tow.o
	$(CC) print.o tow.o -o printArr

main.o: main.cpp
	$(CC) $(CFLAGS) main.cpp -o main.o

mtrx.o: mtrx.cpp
	$(CC) $(CFLAGS) mtrx.cpp -o mtrx.o

tow.o: tow.cpp
	$(CC) $(CFLAGS) tow.cpp -o tow.o

print.o: printer.cpp
	$(CC) $(CFLAGS) printer.cpp -o print.o

clean:
	rm -rf *.o
```

***Приложение 7.*** `offset_test.sh`
*тест на наименьшее смещение, способное вызвать пробуксовку кэша*

```
#!/bin/bash

echo -e "offset test\n"
echo "making the program"
make
if [[ $? -ne 0 ]]; then
 exit
fi
echo -e "the program is ready!\n"

i=6
offset=$(( 1 << i >> 2 ))
size=$(( offset << 3 ))
echo -e "#\tsize\toffset\t1try\t2try\t3try"
           >offset-report.txt
for (( ; i < 24; i++ ))
do
 echo -n -e "$i\t" | tee -a offset-report.txt
 ./associativity $size $offset 100 >>offset-report.txt
    if [[ $? -ne 0 ]];
        then echo "out of memory with size $size ints"
        exit
    fi
 offset=$(( offset << 1 ))
    size=$(( size << 1 ))
done

echo -e "\nall done!"
```

## ***Приложение 8.*** `test.sh`
## *тест на ассоциативность кэша*

```
#!/bin/bash
REPORT_FILE="report.txt"

if [[ $# -ne 1 ]]; then
    echo "arguments count must be equal to 1"
    exit
fi

echo -e "main test\n"
echo "making the program"
make
if [[ $? -ne 0 ]]; then
 exit
fi
echo -e "the program is ready!\n"

offset=$(( 1 << $1 ))
echo -e "#\tsize\toffset\t1try\t2try\t3try" >$REPORT_FILE
for (( i = 1; i < 41; i++ )) do
 echo "$i"
    echo -n -e "$i\t" >>$REPORT_FILE
 ./associativity $(( offset * i )) $offset 100
           >>$REPORT_FILE
 if [[ $? -ne 0 ]]; then
      echo "error on ($offset, $i)"
      exit
 fi
done

echo -e "\nall done!"
```

***Приложение 9.*** *offset_report.txt*
*протокол поиска наименьшего смещения, способного вызвать пробуксовку кэша*

```
#size offset    1try 2try 3try
6     128  16   194  194  194
7     256  32   194  186  186
8     512  64   186  186  219
9     1024 128  186  186  186
10    2048 256  188  186  186
11    4096 512  194  197  194
12    8192 1024 830  804  801
13    16384     2048 812  803  803
14    32768     4096 805  852  852
15    65536     8192 281474976645915     796  808
16    131072    16384     796  797  806
17    262144    32768     815  816  817
18    524288    65536     813  816  813
19    1048576   131072    1226 17592186041509 1192
20    2097152   262144    814  8796093020972  811
21    4194304   524288    796  4398046510876  4398046510875
22    8388608   1048576   164  2199023255204  164
23    16777216  2097152   34   48   47
```

***Приложение 10.*** *report.txt*
*протокол теста на степень ассоциативности кэша*

| # | size | offset | 1try | 2try | 3try |
|---|---|---|---|---|---|
| 1 | 1024 | 1024 | 200 | 203 | 200 |
| 2 | 2048 | 1024 | 186 | 186 | 186 |
| 3 | 3072 | 1024 | 186 | 186 | 200 |
| 4 | 4096 | 1024 | 186 | 194 | 186 |
| 5 | 5120 | 1024 | 186 | 186 | 186 |
| 6 | 6144 | 1024 | 188 | 186 | 186 |
| 7 | 7168 | 1024 | 269 | 262 | 270 |
| 8 | 8192 | 1024 | 799 | 858 | 830 |
| 9 | 9216 | 1024 | 829 | 822 | 800 |
| 10 | 10240 | 1024 | 798 | 794 | 795 |
| 11 | 11264 | 1024 | 805 | 850 | 830 |
| 12 | 12288 | 1024 | 801 | 800 | 798 |
| 13 | 13312 | 1024 | 800 | 834 | 830 |
| 14 | 14336 | 1024 | 799 | 803 | 800 |
| 15 | 15360 | 1024 | 794 | 794 | 798 |
| 16 | 16384 | 1024 | 826 | 824 | 807 |
| 17 | 17408 | 1024 | 798 | 832 | 824 |
| 18 | 18432 | 1024 | 811 | 825 | 824 |
| 19 | 19456 | 1024 | 804 | 811 | 814 |
| 20 | 20480 | 1024 | 797 | 794 | 794 |
| 21 | 21504 | 1024 | 806 | 804 | 813 |
| 22 | 22528 | 1024 | 803 | 821 | 833 |
| 23 | 23552 | 1024 | 803 | 816 | 831 |
| 24 | 24576 | 1024 | 818 | 820 | 805 |
| 25 | 25600 | 1024 | 794 | 794 | 794 |
| 26 | 26624 | 1024 | 807 | 824 | 818 |
| 27 | 27648 | 1024 | 838 | 806 | 832 |
| 28 | 28672 | 1024 | 813 | 823 | 809 |
| 29 | 29696 | 1024 | 830 | 800 | 832 |

| | | | | | |
|---|---|---|---|---|---|
| 30 | 30720 | 1024 | 797 | 795 | 794 |
| 31 | 31744 | 1024 | 815 | 808 | 825 |
| 32 | 32768 | 1024 | 823 | 805 | 822 |
| 33 | 33792 | 1024 | 1136 | 1122 | 1127 |
| 34 | 34816 | 1024 | 1114 | 1128 | 1129 |
| 35 | 35840 | 1024 | 1110 | 1103 | 1103 |
| 36 | 36864 | 1024 | 1127 | 1121 | 1134 |
| 37 | 37888 | 1024 | 1133 | 1138 | 1125 |
| 38 | 38912 | 1024 | 1127 | 1133 | 1132 |
| 39 | 39936 | 1024 | 1144 | 1123 | 1136 |
| 40 | 40960 | 1024 | 1103 | 1102 | 1103 |