###### 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»

студента 2 курса, группы 21206

**Балашова Вячеслава Вадимовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2022

**Содержание**

Цель3

Задание3

Описание работы4

Заключение6

Приложение 1. Исходный код файла MulMatrices.h7

Приложение 2. Исходный код программы, вычисляющей время обращения к памяти 8

**Цель**

Экспериментальное определение степени ассоциативности кэш-памяти.

**Задание**

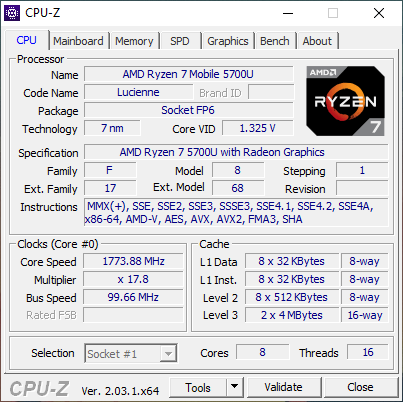
1. Написать программу, выполняющую обход памяти в соответствии с заданием.
2. Измерить среднее время доступа к одному элементу массива (в тактах процессора) для разного числа фрагментов: от 1 до 32. Построить график зависимости времени от числа фрагментов.
3. По полученному графику определить степень ассоциативности кэш-памяти, сравнить с реальными характеристиками исследуемого процессора.
4. Составить отчет по практической работе.

**Описание работы**

Ход выполнения работы:

1. Был создан файл MulMatrices.h (См. Приложение 1), содержащий в себе две функции: зануление матрицы и перемножение матриц. Обе функции были взяты из практики про векторизацию и слегка изменены (добавлен еще один аргумент – size – размер матрицы).
2. На основе размеров кэш-памяти процессора был выбран шаг (offset) = 16 МБ. Данное число кратно размеру кэш-памяти каждого уровня
3. Была написана программа на языке программирования C++, выполняющая обход массива, заполненного с целью получения пробуксовки кэш-памяти (см. Приложение 2). Обходы формировались в зависимости от количества фрагментов (0 <n <101), количества элементов в этих фрагментах:
   1. i <n – все элементы фрагмента переходят в соответствующие элементы следующего фрагмента.
   2. i =n – все элементы фрагмента переходят в соответствующие элементы первого фрагмента, но с циклическим сдвигом в 1.
4. После «прогрева» процессора с помощью перемножения матриц, для каждого количества фрагментов замеряется среднее время обращения к одному элементу массива. Данный замер делается 6 раз, и из всех этих замеров берется минимальное по времени значение для каждого числа фрагментов.
5. Программа компилировалась на уровне оптимизации O1, поэтому в программе присутствуют операции, не сильно влияющие на производительность и препятствующие удалению внутреннего цикла, осуществляющего обход.
6. На основе полученных данных был построен график зависимости времени обращения к элементу в зависимости от количества фрагментов. 
7. На графике видны замедления. Количество фрагментов, на которых происходят замедления, соответствуют степени ассоциативности кэш-памяти соответствующих уровней.
8. Была определена ассоциативность кэш-памяти разных уровней с помощью утилиты CPU-Z, отображающей характеристики процессора. Полученные результаты были сравнены.

Рис. 1. Вывод программы CPU-Z



**Заключение**

В ходе выполнения лабораторной работы были экспериментальным методом получены степень ассоциативности кэш-памяти процессора.

С помощью анализа графика были получены значения:

1. 8 – степень ассоциативности кэш-памяти первого и второго уровней.
2. 32 – степень ассоциативности кэш-памяти третьего уровня.

С помощью утилиты CPU-Z были получены значения:

1. 8 – степень ассоциативности кэш-памяти первого и второго уровней.
2. 16 – степень ассоциативности кэш-памяти третьего уровня.

Для разных способов определения ассоциативности кэш-памяти процессора Ryzen 7 5700U были получены одинаковые значения степени ассоциативности кэш-памяти первого и второго уровней, и разные значения для степени ассоциативности кэш-памяти третьего уровня: 16 против 32.

Такой разброс получился из-за более сложного устройства кэш-памяти третьего уровня.

**Приложение 1. Исходный код файла MulMatrices.h.**

#pragma once

#include <x86intrin.h>

inline void AVX2\_FillZero(float \* Matrix, const size\_t size)

{

auto reg0 = \_mm256\_setzero\_ps();

for (auto i = Matrix; i < Matrix + size \* size; i += 8)

{

\_mm256\_store\_ps(i, reg0);

}

}

inline void AVX2\_MulMatrices(const float \* A, const float \* B, float \* Res, const size\_t size)

{

AVX2\_FillZero(Res, size);

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

{

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

{

float buf[8];

std::fill(buf, buf + 8, A[i \* size + k]);

auto reg0 = \_mm256\_load\_ps(buf);

for (int j = 0; j < size; j+=8)

{

auto reg1 = \_mm256\_load\_ps(B + (k \* size) + j);

auto reg2 = \_mm256\_load\_ps(Res + (i \* size) + j);

reg1 = \_mm256\_mul\_ps(reg0, reg1);

reg2 = \_mm256\_add\_ps(reg2, reg1);

\_mm256\_store\_ps(Res + (i \* size) + j, reg2);

}

}

}

}

**Приложение 2. Исходный код программы, вычисляющей время обращения к памяти.**

#include <iostream>

#include <x86intrin.h>

#include "MulMatrices.h"

constexpr size\_t LEVEL1\_BANK\_SIZE = 32 \* 1024 / 8;

constexpr size\_t LEVEL2\_BANK\_SIZE = 512 \* 1024 / 8;

constexpr size\_t LEVEL3\_BANK\_SIZE = 4 \* 1024 \* 1024 / 16;

constexpr size\_t CACHE\_LINE\_SIZE = 64;

constexpr size\_t OFFSET = 16 \* 1024 \* 1024;

constexpr size\_t MAX\_NUM\_OF\_FRAGMENTS = 100;

constexpr size\_t SIZE\_T\_OFFSET = OFFSET / sizeof(size\_t);

constexpr size\_t NUM\_OF\_ITERATIONS = 10;

constexpr size\_t SIZE\_OF\_CHECKING\_ARRAY = 64;

static size\_t buf = 0;

inline void FillArray(size\_t \* array, size\_t size)

{

for (size\_t i = 0; i < size - SIZE\_T\_OFFSET; i += SIZE\_T\_OFFSET)

{

for (size\_t j = 0; j < SIZE\_OF\_CHECKING\_ARRAY; j++)

{

array[i + j] = i + j + SIZE\_T\_OFFSET;

}

}

for (size\_t j = 0; j < SIZE\_OF\_CHECKING\_ARRAY; j++)

{

array[size - SIZE\_T\_OFFSET + j] = (j + 1) % SIZE\_OF\_CHECKING\_ARRAY;

}

}

int main()

{

std::cerr << sizeof(size\_t) << std::endl;

auto \* matrix1 = new float[1024 \* 1024];

auto \* matrix2 = new float[1024 \* 1024];

auto \* matrix3 = new float[1024 \* 1024];

for (size\_t i = 0; i < 10; i++)

{

AVX2\_MulMatrices(matrix1, matrix2, matrix3, 1024);

}

std::cerr << matrix3[0];

auto \* array = new size\_t[MAX\_NUM\_OF\_FRAGMENTS \* SIZE\_T\_OFFSET];

for (size\_t i = 1; i <= MAX\_NUM\_OF\_FRAGMENTS; i++)

{

FillArray(array, i \* SIZE\_T\_OFFSET);

long double result = UINT64\_MAX;

for (size\_t j = 0; j < 6; j++)

{

size\_t k = 0, counter = 0;

size\_t start = \_\_rdtsc();

while (counter < NUM\_OF\_ITERATIONS)

{

k = array[k];

counter += !k;

}

size\_t end = \_\_rdtsc();

buf = counter / i;

result = std::min(result, static\_cast <long double> (end - start) /

(static\_cast<long double> (SIZE\_OF\_CHECKING\_ARRAY) \* i \* NUM\_OF\_ITERATIONS));

std::cerr << buf;

}

std::cout << i << "; " << result << std::endl;

}

std::cerr << std::endl;

delete[] array;

delete[] matrix1;

delete[] matrix2;

delete[] matrix3;

return 0;

}