i.МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ii.ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

iii.НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

iv.Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«ИЗМЕРЕНИЕ СТЕПЕНИ АССОЦИАТИВНОСТИ КЭШ-ПАМЯТИ»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2022

СОДЕРЖАНИЕ

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. Отчет должен содержать следующее.

a. Титульный лист.
b. Цель практической работы.
c. Параметры теста: размер фрагментов, величина смещения.
d. График зависимости среднего времени доступа к элементу массива от числа фрагментов.
e. Оценку степени ассоциативности различных уровней кэш-памяти согласно выполненным вычислительным экспериментам.
f. Реальные значения степеней ассоциативности различных уровней кэш-памяти процессора, подкрепленные доказательствами (скриншоты из программ типа CPU-Z, файлы операционной системы, куски официальной документации по процессору и т.д.)
g. Полный компилируемый листинг реализованной программы и команды для ее компиляции.
h. Вывод по результатам работы

ОПИСАНИЕ РАБОТЫ

1. Была написана программа для экспериментального определения степени ассоциативности кэшпамятию. В программе организуется обход данных в памяти, который вызывает «буксование» кэш-памяти. Для этого организовывается доступ в память по адресам, отображаемым на одно и то же множество в кэш-памяти.

2. Было измерено среднее время доступа к одному элементы массива (размер фрагмента был выбран 8 МБ, смещение - 16 МБ)

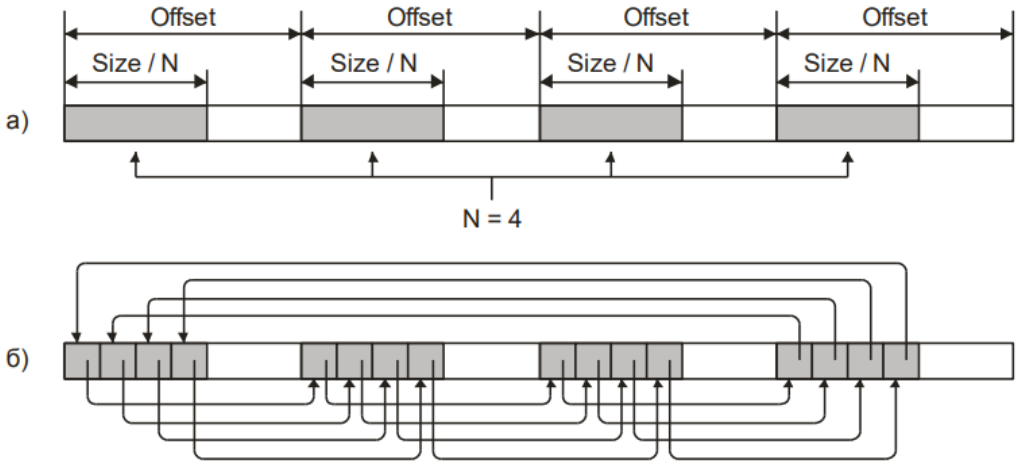

Рис. 1. Схема расположения в памяти фрагментов данных для обхода (а) и порядок обхода элементов (б)

3. Был построен график зависимости времени чтения элемента массива от числа фрагментов

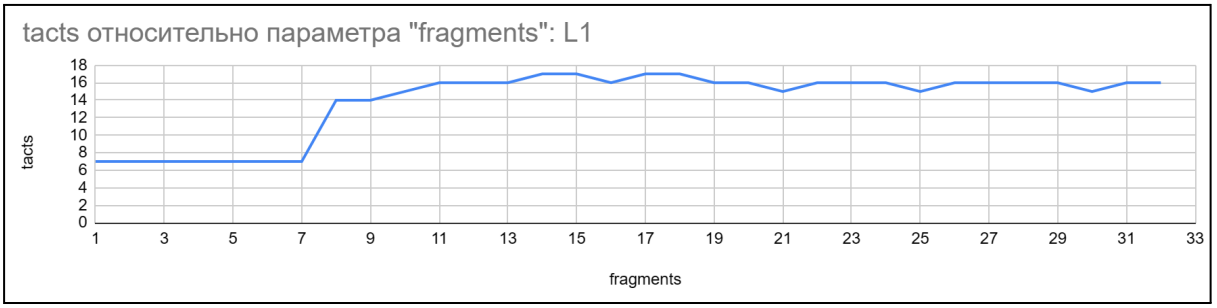

Рис. 2. График зависимости времени чтения элемента массива от числа фрагментов (block size = 32 kb)

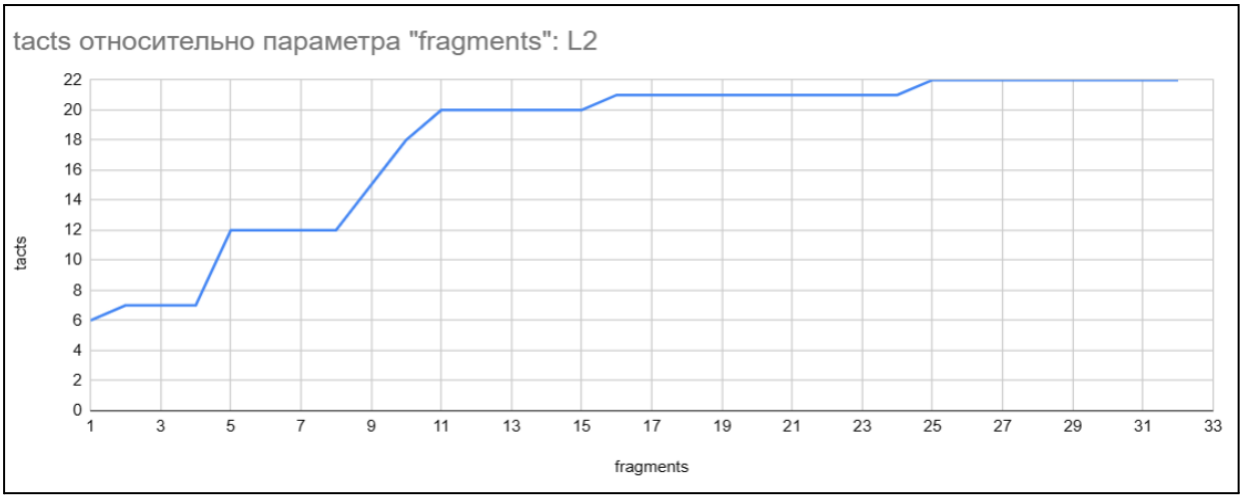

Рис. 3. График зависимости времени чтения элемента массива от числа фрагментов (block size = 256 kb)

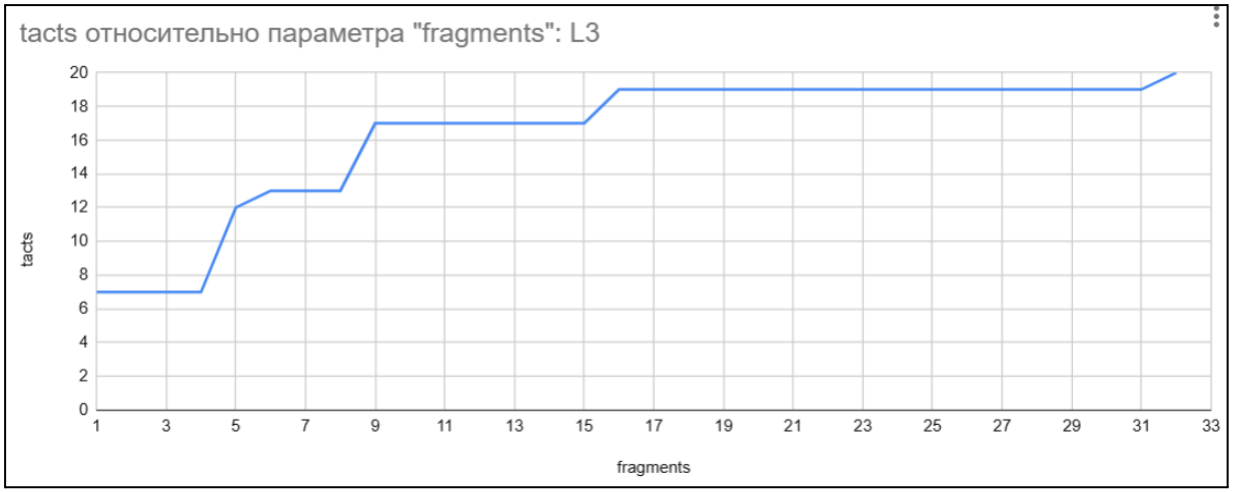

Рис. 4. График зависимости времени чтения элемента массива от числа фрагментов (block size =8Mb)

4. С помощью установленной прогрыммы CPU-Z была получена информация о кэше.

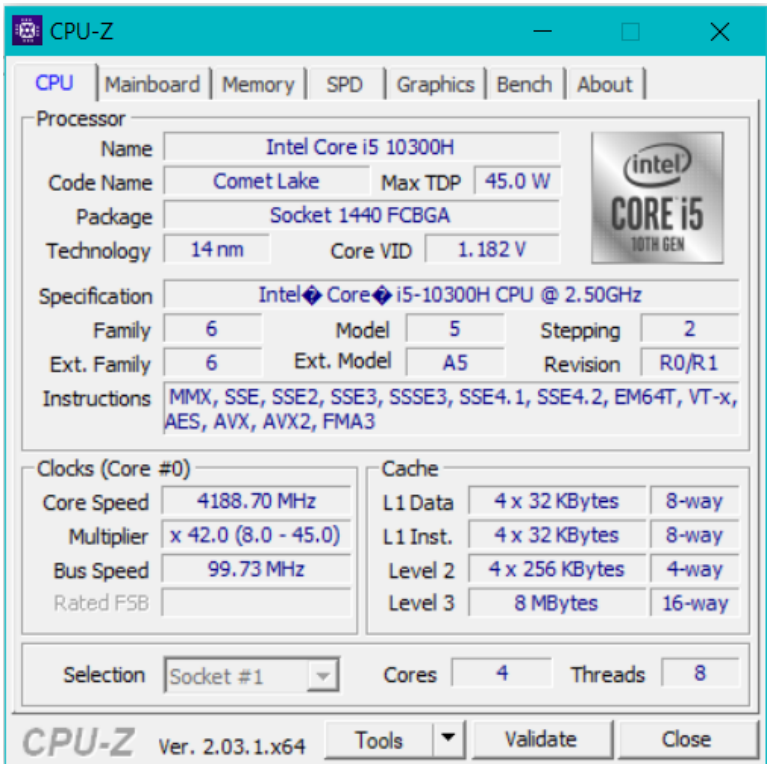

Рис. 5. Скриншот работы программы CPU-Z

5. Были сделаны выводы на полученных данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было экспериментальное определена степень ассоциативности кэш-памяти.

По результатам полученных данных можно сделать вывод: на графике есть несколько существенных скачков, при количестве фрагментов равных 4, 8, и 16, что соответсвует данным, полученным с помощью приложения CPU-Z (степени ассоциативности кэшей L1 равна 8, L2 равна 4, а степень ассоциативности L3 кэша равна 16).

*Запуск программы осуществлялся на устрйстве с процессором Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz   2.50 GHz.

ПРИЛОЖЕНИЕ

Приложение 1. Полный листинг программы на C++

```
1.  #include <iostream>
2.  #include <climits>
3.  #include <iomanip>
4.  #include <x86intrin.h>
5.
6.  const unsigned int N = 200000000;
7.  const unsigned int RUN_TIMES = 3;
8.  const unsigned int FRAGMENTS_COUNT = 32;
9.
10. void initArray(unsigned int *array, unsigned int fragments, size_t offset, size_t
    size);
11. unsigned long run(unsigned int const *array);
12. void countTime(unsigned int *array, unsigned int fragments, int offset, int size);
13.
14. int main(void){
15.
16.     auto *array = new unsigned int[N];
17.     // unsigned int blockSize = 32 * 1024; // 32 kb
18.     // unsigned int blockSize = 256 * 1024 * 1024; // 256 kb
19.     unsigned int blockSize = 8 * 1024 * 1024; // 8 MB
20.     unsigned int offset = blockSize;
21.
22.     for(int fragments = 1; fragments <= FRAGMENTS_COUNT; fragments++)
23.     {
24.         countTime(array, fragments, offset / sizeof(int), blockSize / sizeof(int));
25.     }
26.
27.     delete[] array;
28.     return 0;
29. }
30.
31. void initArray(unsigned int *array, unsigned int fragments, size_t offset, size_t
    size)
32. {
33.     size_t j = 1;
34.     size_t i = 0;
35.
36.     for(; i < size; i++)
37.     {
38.         for(j = 1; j < fragments; j++)
39.             array[i + (j - 1) * offset] = i + j * offset;
40.
41.         array[i + (j - 1) * offset] = i + 1;
42.     }
43.
44.     array[i - 1 + (j - 1) * offset] = 0;
45. }
46.
47. unsigned long run(unsigned int const *array)
48. {
```

```
49.     unsigned long long minTime = ULLONG_MAX;
50.
51.     for(size_t j = 0; j < RUN_TIMES; j++)
52.     {
53.         unsigned long long startTime = __rdtsc();
54.         for(size_t k = 0, i = 0; i < N; i++){
55.             k = array[k];
56.         }
57.         unsigned long long endTime = __rdtsc();
58.
59.         if (minTime > endTime - startTime) minTime = endTime - startTime;
60.     }
61.
62.     return minTime / N;
63. }
64.
65. void countTime(unsigned int *array, unsigned int fragments, int offset, int size)
66. {
67.     initArray(array, fragments, offset, size);
68.     std::cout << std::left << std::setw(2)
69.                         << fragments << " fragments "
70.                         << std::setw(2)
71.                         << run(array) << " tacts"
72.                         << std::endl;
73. }
```