Федеральное государственное бюджетное образовательное учреждение высшего образования. «Национально исследовательский университет «Московский энергетический институт»

Кафедра ВМСС

Лабораторная работа №6

ИССЛЕДОВАНИЕ ПРИНЦИПОВ РАБОТЫ КЭШ-ПАМЯТИ

Курс: Вычислительные системы

Группа: А-08-19

Выполнил: Кретов Н.В.

Проверил: Карпов А.В.

Москва 2023 г.

**Домашняя подготовка**

Цель работы: изучение основных принципов работы кэш-памяти, знакомство с алгоритмами замещения строк кэш-памяти, рассмотрение понятия ассоциативности кэш-памяти.

Таблица 1

Вариант для выполнения домашней подготовки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ Бригады** | **Разрядность системы** | **Размер кэш-строки**  **(в байтах)** | **Кол-во кэш-строк** | **Степень ассоциативности** | **Последовательность байт** | **Размер ОЗУ** |
| 11 | 16 | 64 | 4 | 8 | CAB, 3E80, 3E00, 3E, 3BD0, FF | 16 КБ |

**1.** Изучить соответствующие разделы лекционного курса и описания лабораторных работ.

**2.** Исследование кэша прямого отображения. **Степень ассоциативности** для данного пункта **игнорировать**. Рассчитать **полезный** размер кэш-памяти исходя из данных, представленных в таблице 1. Схематично представить структуру заданной кэш-памяти (на схеме должны быть отражены: **размер** и **количество кэш-строк**, **тег**, соответствующий каждой кэш-строке). Определить возможное расположение заданной в таблице 1 последовательности байт в кэш-памяти при прямом отображении ОЗУ в кэш-память. Рассчитать вероятность коллизии в случае обращения к ячейкам ОЗУ, адреса которых отличаются на N, где N – полезный размер кэш-памяти.

Полезный размер кэш памяти: (байт).

Теги строк представим в виде таблицы 2.

Таблица 2

Теги кэш-строк

|  |  |  |
| --- | --- | --- |
| **№ кэш-строки** | **Тег** | |
| **Десятичная СС** | **Шестнадцатеричная СС** |
| 1 | 0 | 00 |
| 2 | 64 | 40 |
| 3 | 128 | 80 |
| 4 | 192 | С0 |

Пояснение: теги не обязательно перечислять в таблице (но если кэш-строк слишком много, то так удобнее), можно их подписать на схеме.

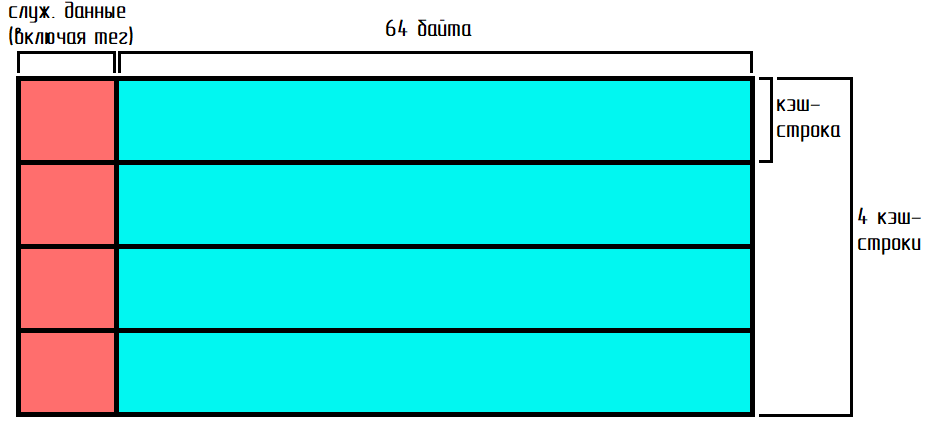


Рис. 1. Структура кэш-памяти

Определим индексы кэш-строк для заданной последовательности байт (см. таблицу 1):. Результаты вычислений занесем в таблицу 3.

Таблица 3

Индексы кэш-строк для последовательности байт

|  |  |  |
| --- | --- | --- |
| **Индекс кэш-строки** | **Адрес байта в ОЗУ** | |
| **Шестнадцатеричная СС** | **Десятичная СС** |
| 2 | CAB | 3243 |
| 2 | 3E80 | 16000 |
| 0 | 3E00 | 15872 |
| 0 | 3E | 62 |
| 3 | 3BD0 | 15312 |
| 3 | FF | 255 |

Пояснение: индекс кэш-строки, в отличии от её номера, считается с 0.

Рассчитаем вероятность коллизии: .

Пояснение: 1) 64, потому что размер кэша/размер ОЗУ = 64; 2) вероятность коллизии столь велика, потому что мы не учитываем алгоритм предсказания обращения к памяти, степень ассоциативности, а также берем наихудший случай: последовательное обращение к тем байтам, которые претендуют на одну и ту же кэш-строку.

1. Исследование понятия ассоциативности. Рассчитать **полезный** размер кэш-памяти исходя из данных, представленных в таблице 1 **с учётом числа банков**. Схематично представить структуру заданной кэш-памяти (на схеме должны быть отражены: **количество банков** и **полезный** **размер** кэш-памяти в банке). Рассчитать **вероятность коллизии** в случае обращения к ячейкам ОЗУ, **адреса** которых **отличаются** **на** **N**, где N – полезный размер кэш-памяти.Сравнить с результатом, полученным в пункте 2.

Полезный размер кэш памяти с учетом числа банков: (байт).

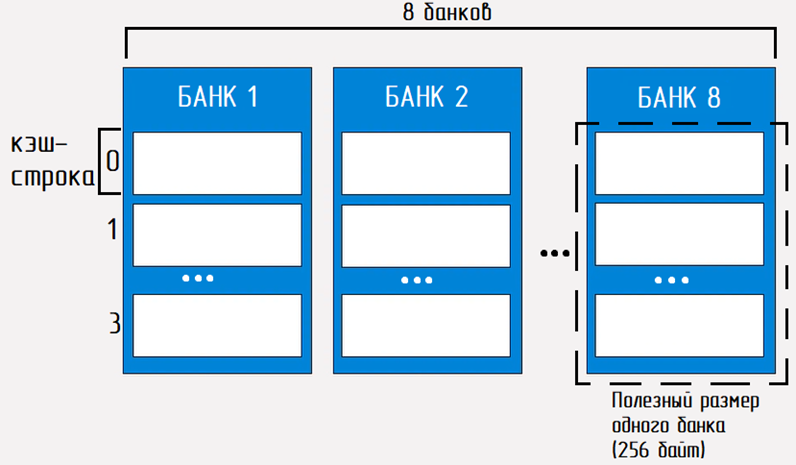


Рис. 2. Структура кэш-памяти

Рассчитаем вероятность коллизии: .

Пояснение: 8, потому что каждая строка ОЗУ может быть отображена в любую из 8 кэш-строк разных банков.  
Вывод: с увеличением степени ассоциативности кэш-памяти увеличивается её полезный размер, также снижается вероятность коллизии (за счет увеличения числа строк кэш-памяти, в которые может быть отражены несколько претендующих на это строк из ОЗУ).

1. Описать процесс **замещения** строк кэш-памяти при последовательном обращении к набору байт из пункта 2 в случае, когда кэш-память уже заполнена данными (лежащими в ОЗУ по адресу от 0 до полезного размера кэш-памяти, **учитывая степень ассоциативности**).

**Для групп А-07, А-12** алгоритм **LRU** (Least Recently Used) замещения строк кэш-памяти.

**Для группы А-08** алгоритм **MFU** (Most Frequently Used) замещения строк кэш-памяти.

Представим начальное состояние кэш-памяти в форме таблицы 4. Пояснение: в ячейках – первые из адресов байтов ОЗУ, которые лежат в указанной кэш-строке указанного банка и кол-во обращений к этим ячейкам (кол-во обращений указано случайным образом).

Таблица 4

Начальное состояние кэш-памяти

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Индекс кэш-строки** | **Банк** | | | | | | | | | | | | | | | |
| **1** | | **2** | | **3** | | **4** | | **5** | | **6** | | **7** | | **8** | |
| 0 | 0 | 1 | 256 | 3 | 512 | 1 | 768 | 1 | 1024 | 5 | 1280 | 1 | 1536 | 2 | 1792 | 4 |
| 1 | 64 | 2 | 320 | 2 | 576 | 2 | 832 | 1 | 1088 | 1 | 1344 | 1 | 1600 | 5 | 1856 | 4 |
| 2 | 128 | 1 | 384 | 1 | 640 | 2 | 896 | 5 | 1152 | 2 | 1408 | 1 | 1664 | 3 | 1920 | 1 |
| 3 | 192 | 4 | 448 | 1 | 704 | 4 | 960 | 2 | 1216 | 5 | 1472 | 2 | 1728 | 2 | 1984 | 1 |

Алгоритм MFU заключается в том, что, при необходимости замещения, замещается та кэш-строка, к которой было совершено наибольшее кол-во обращений за последнее время. В таком случае, при обращении к байу **3243** (**САВ** в 16-ричной СС), будет заменена **2я строка 4го банка**, так как из всех строк с индексом 2 к ней обращались наибольшее кол-во раз (5). Строка с байтом **16000** (**3E80**) заменит **2ю строку 7го банка**, **15872** (**3E00**) – **0ю строку 5го банка**, при обращении к байту **62** (**3E**) никакая из кэш-строк не будет замещена (уже лежит в кэш-памяти), **15312** (**3BD0**) – **3ю строку 5го банка**, **255** (**FF**) – кэш-строка замещена не будет.

Таблица 4

Конечное состояние кэш-памяти

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Индекс кэш-строки** | **Банк** | | | | | | | | | | | | | | | |
| **1** | | **2** | | **3** | | **4** | | **5** | | **6** | | **7** | | **8** | |
| 0 | 0 | 1 | 256 | 3 | 512 | 1 | 768 | 1 | 15872 | 1 | 1280 | 1 | 1536 | 2 | 1792 | 4 |
| 1 | 64 | 3 | 320 | 2 | 576 | 2 | 832 | 1 | 1088 | 1 | 1344 | 1 | 1600 | 5 | 1856 | 4 |
| 2 | 128 | 1 | 384 | 1 | 640 | 2 | 3200 | 1 | 1152 | 2 | 1408 | 1 | 16000 | 1 | 1920 | 1 |
| 3 | 192 | 5 | 448 | 1 | 704 | 4 | 960 | 2 | 15296 | 1 | 1472 | 2 | 1728 | 2 | 1984 | 1 |

**Лабораторное задание**

1. Написать программу на языке C, позволяющую отследить изменение в скорости отклика получения данных при явлении **thrashing**’а (**пробуксовке кэша**).
2. На основе данных, полученных в пункте 1, **построить график**, **определить** по нему **размер** кэша **L1** и **L2**.

Проанализировать полученные результаты и **сделать вывод**.

Программа:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <iostream>

#include <Windows.h>

using namespace std;

#define T char

#define MAX\_S 0x1000000

#define L 101

volatile T A[MAX\_S];

int m\_rand[0xFFFFFF];

int main (){

LARGE\_INTEGER freq; LARGE\_INTEGER time1; LARGE\_INTEGER time2;

QueryPerformanceFrequency(&freq);

memset ((void\*)A, 0, sizeof (A));

srand(time(NULL));

int v, M;

register int i, j, k, m, x;

for (k = 1024; k < MAX\_S;) {

M = k / L;

printf("%g\t", (k+M\*4)/(1024.\*1024));

for (i = 0; i < M; i++) m\_rand[i] = L \* i;

for (i = 0; i < M/4; i++) {

j = rand() % M;

x = rand() % M;

m = m\_rand[j];

m\_rand[j] = m\_rand[i];

m\_rand[i] = m;

}

if (k < 100\*1024) j = 1024;

else if (k < 300\*1024) j = 128;

else j = 32;

QueryPerformanceCounter(&time1);

for (i = 0; i < j; i++) {

for (m = 0; m < L; m++) {

for (x = 0; x < M; x++){

v = A[ m\_rand[x] + m ];

}

}

}

QueryPerformanceCounter(&time2);

time2.QuadPart -= time1.QuadPart;

double span = (double) time2.QuadPart / freq.QuadPart;

printf ("%g\n",1000000000. \* span/(double)(L\*M\*j));

if (k > 100\*1024) k += k/16;

else k += 4\*1024;

}

return 0;

}

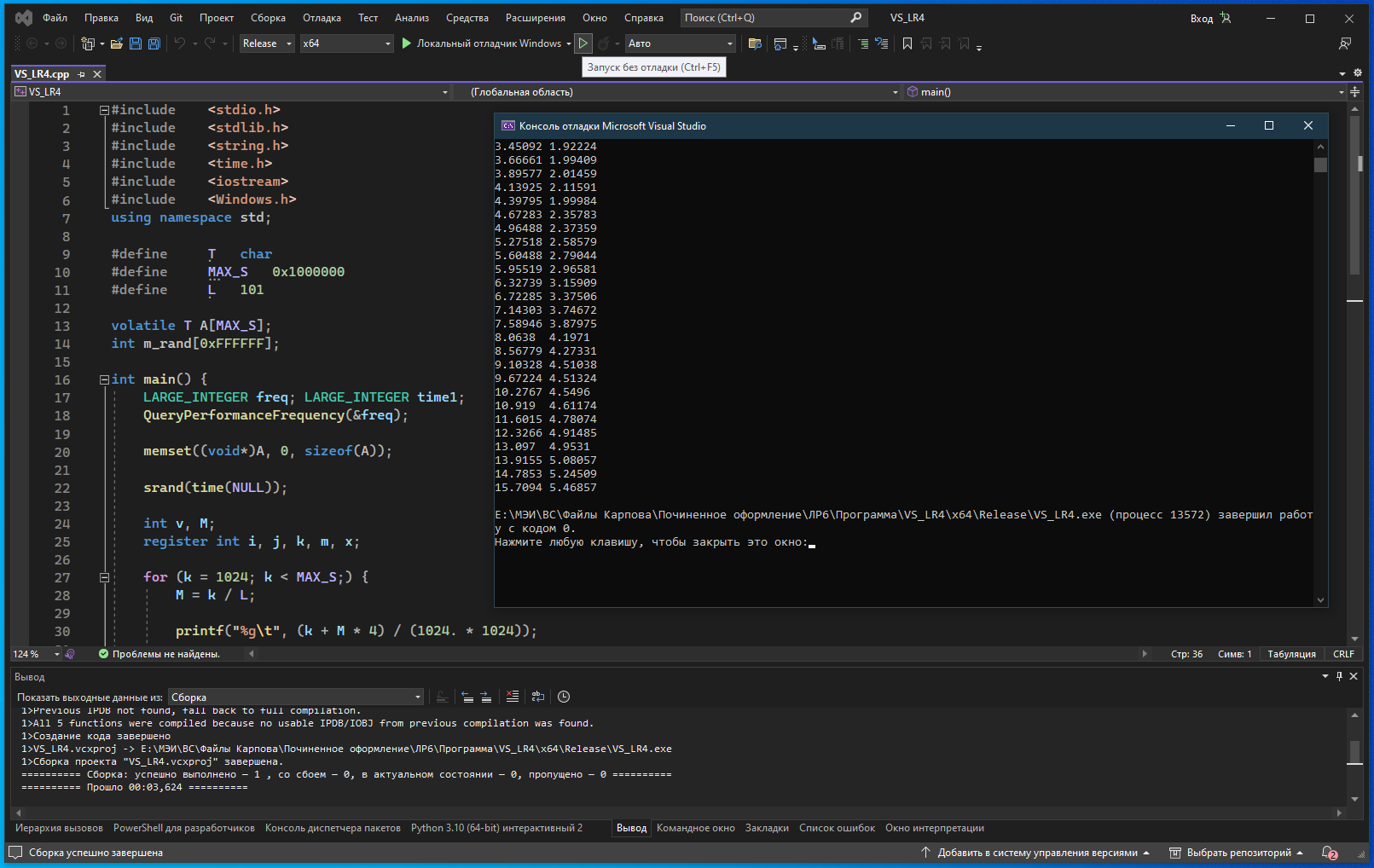


Рис. 3. Результат работы программы

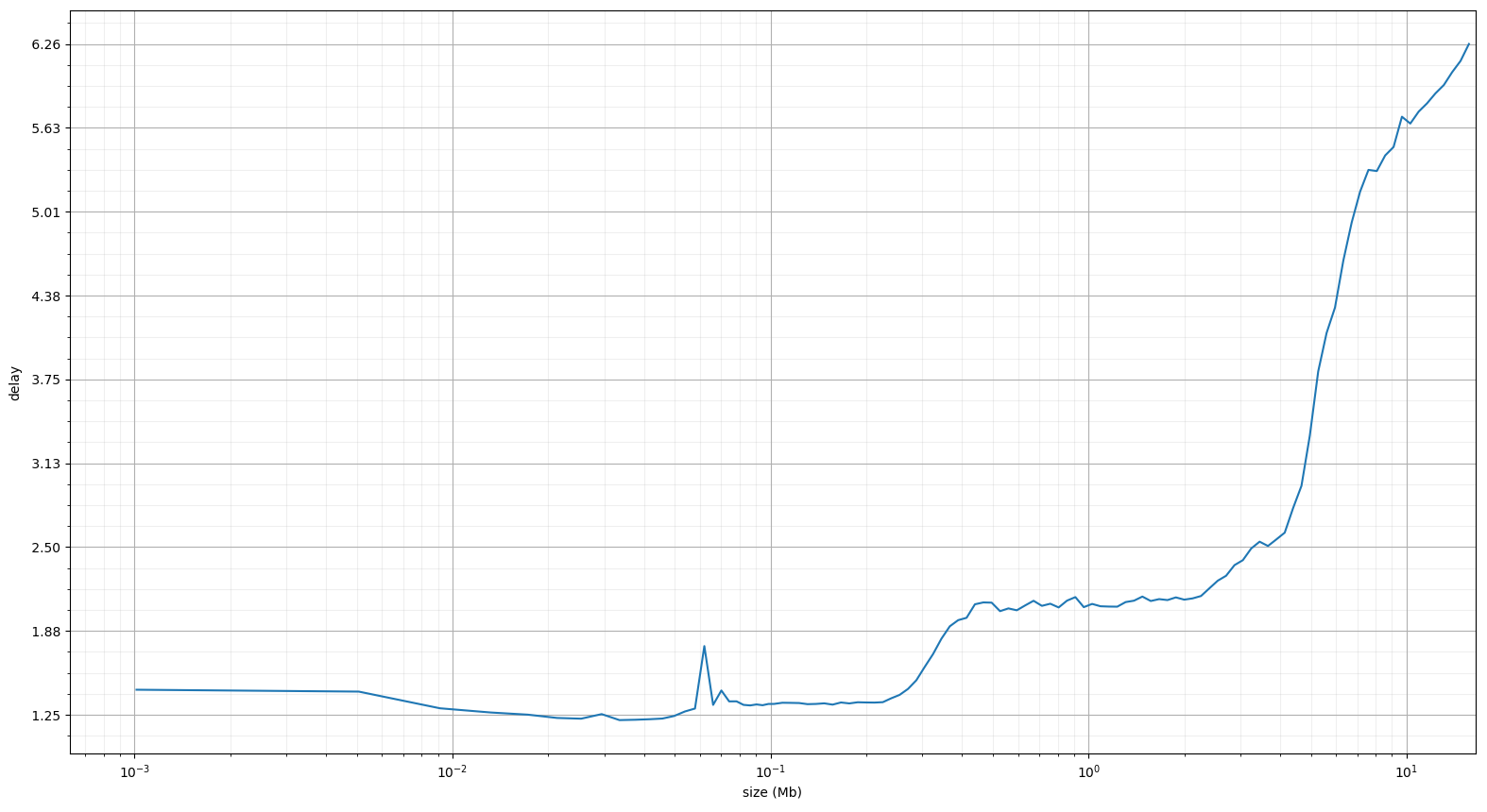


Рис. 4. График по результатам работы программы (в логарифмическом масштабе)

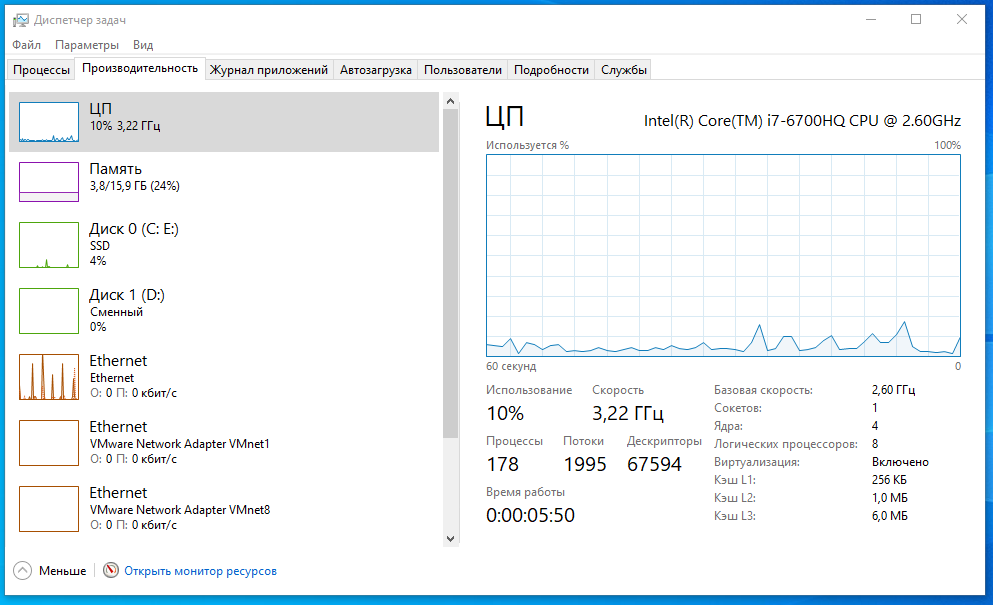


Рис. 5. Реальные размеры кэша L1 и L2

Вывод: скачки на графике иллюстрируют переход от кэша более низкого уровня к кэшу более высокого уровня. По графику видно, что первый скачок происходит примерно при 0,06 МБ или 61,44 КБ. Умножим эту величину на количество ядер (см. Рис. 5) и получим размер кэша L1: 61,44\*4=245,76 КБ. Следующий скачок наблюдается при 0,2 МБ. Также умножим на кол-во ядер и получим размер кэша L2: 0,2\*4=0,8 МБ. Сравнивая результаты с реальными размерами кэша, можно заметить неплохую точность программного нахождения объема кэш-памяти.