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 ЛАБОРАТОРНОЙ РАБОТЫ

«ВЛИЯНИЕ КЭШ-ПАМЯТИ НА ВРЕМЯ ОБРАБОТКИ МАССИВОВ»

студента 2 курса, 21202 группы

Куращенко Льва Владиславовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Перепелкин В.А.

Новосибирск 2019

1. **Цель**

Изучить принцип работы работа кэш-память процессора и на примере обхода элементов массива в разном порядке выяснить размер разных уровней кэш-памяти на конкретном ПК.

2. **Задание**

На основе разных способов обхода циклического массива: прямого, обратного и случайного, сделать среднюю оценку количества тактов для обращения к одному элементу массива. На основе полученных данных выяснить приблизительный размер кэша процессора.

3. **Описание обходов**

Используются следующие обходы массива:

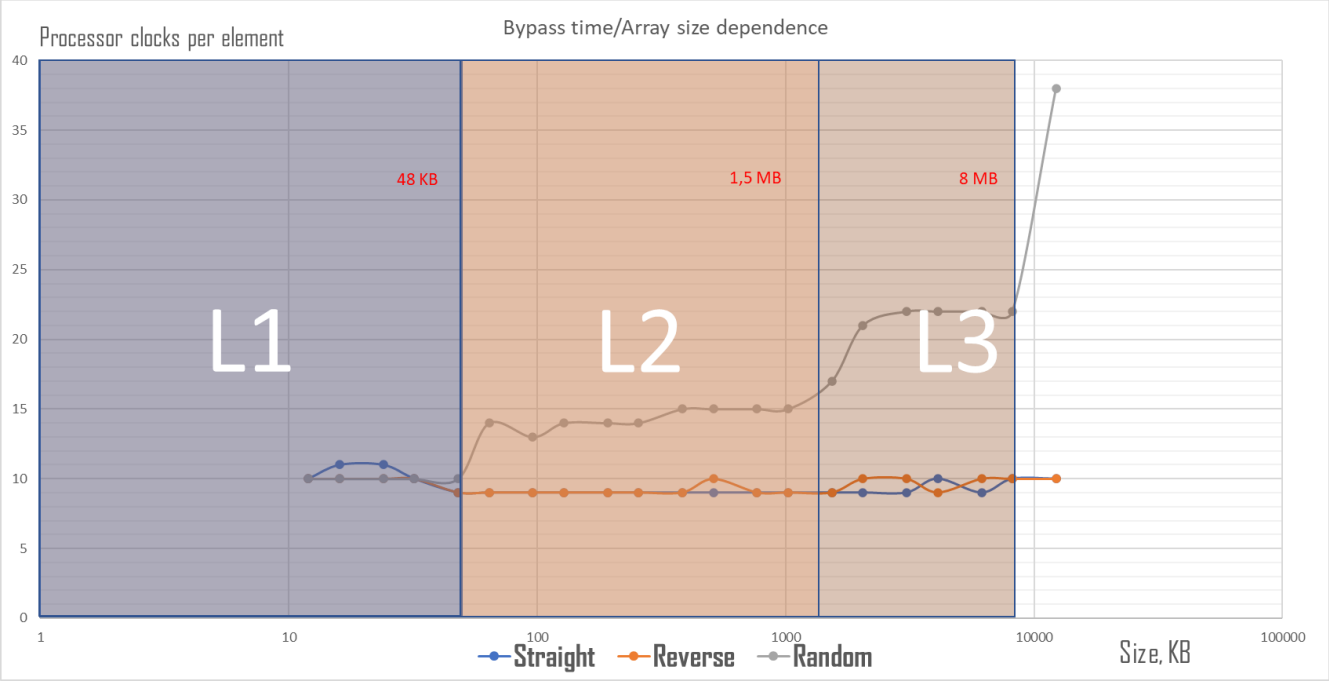
- Прямой;
- Обратный;
- Случайный.

Прямой обход подразумевает, что значением ячейки массива с индексом i будет i+1, т.е. индекс следующей соседней ячейки массива, в случае последнего элемента следующим элементом будет самый первый. Таким образом мы получаем следующий индекс и так обходим массив несколько раз.

Обратный обход – прямой обход, но с условием, что теперь обход идёт с конца массива линейным порядком.

Случайный обход подразумевает, что значением ячейки массива с индексом i будет некоторый индекс, который невозможно предугадать, но гарантируется, что индекс не выйдет за пределы массива. Тем самым, ходя по разным индексам, мы полностью обходим массив.

4. График роста тактов обращения



5. Листинг программы

```
#include <iostream>
#include <set>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <fstream>

#define MIN_SIZE (1 * 1024)
#define MAX_SIZE (3 * 1024 * 1024)
//12 мб
/*
 * How to execute:
 *>g++ Cache.cpp -o cache -O1
 *>./cache >log.txt
 */


using namespace std;

void fillStraight(unsigned int *array, unsigned int n)
{
    for(unsigned int i = 0; i < n - 1; ++i) {
        array[i] = i + 1;
    }
    array[n - 1] = 0;
}

void fillReverse(unsigned int *array, unsigned int n)
{
    for(unsigned int i = n - 1; i > 0; --i) {
        array[i] = i - 1;
    }
    array[0] = n - 1;
}


void fillRandom(unsigned int *array, unsigned int n)
{
    srand(time(NULL));
```

```
    set<unsigned int> set;
    for(unsigned int i = 0; i < n; ++i)
        set.insert(i);

    unsigned int index = rand() % set.size();
    unsigned int prev = *next(set.begin(), index);
    unsigned int start = prev;

    for(; set.size() > 1;)
    {
        set.erase(next(set.begin(), index));
        index = rand() % set.size();
        array[prev] = *next(set.begin(), index);
        prev = *next(set.begin(), index);
    }
    array[*set.begin()] = start;
}

bool step = false;
unsigned int step0 = 0;

void updateSize(unsigned int& size)
{
    if(!step)
        step0 = size / 2;
    size += step0;
    step = !step;
}

unsigned int m = 0;
void bypass(const unsigned int *array, unsigned int n, unsigned int& k)
{
    for(unsigned int j = 0; j < n * k; ++j)
        m = array[m];
}

union ticks
{
    long long t64;
    struct s32
    {
        long th, tl;
    } t32;
} start0, end0;

void writearrays(){
    for (unsigned int n = MIN_SIZE; n <= MAX_SIZE; updateSize(n)) {
        auto s = std::to_string(n/256);
        auto output = "array" +s+".txt";
        cout<<output<<endl;
        std::ofstream outfile (output);
        auto * array = new uint32_t[n];
        fillRandom(array, n);
        for (int i = 0; i < n; ++i) {
            outfile <<array[i];
            outfile<<endl;
        }
        outfile.close();
    }
}

void readarray(std::ifstream& f, unsigned int * array, unsigned int n){
    for (unsigned int i = 0; i < n; i++)
        f>>array[i];
}

int main(){
```

```
//    //До 12 мбайт
    unsigned long long Time = 0;
    std::cout << std::setw(10) << std::left << "Size, KB"
                              << std::setw(10) << std::left << "Straight"
                              << std::setw(10) << std::left << "Reverse"
              << std::setw(10) << std::left << "Random" << std::endl;
    for (unsigned int n = MIN_SIZE; n <= MAX_SIZE; updateSize(n)) {
        cout << std::setw(10) << (double) n / 256;
        unsigned int *array = new uint32_t[n];
        auto s = std::to_string(n / 256);
        auto input = "array" + s + ".txt";
        std::ifstream input_file(input);
        unsigned int k = 1000;
        fillStraight(array, n);
        for (unsigned int j = 0; j < n; ++j)
            m = array[m];
        asm("rdtsc\n":"=a"(start0.t32.th), "=d"(start0.t32.tl));
        bypass(array, n, k);
        asm("rdtsc\n":"=a"(end0.t32.th), "=d"(end0.t32.tl));
        Time = end0.t64 - start0.t64;
        cout << std::setw(10) << Time / (k * n);

        fillReverse(array, n);
        for (unsigned int j = 0; j < n; ++j)
            m = array[m];
        asm("rdtsc\n":"=a"(start0.t32.th), "=d"(start0.t32.tl));
        bypass(array, n, k);
        asm("rdtsc\n":"=a"(end0.t32.th), "=d"(end0.t32.tl));
        unsigned long long Time = end0.t64 - start0.t64;
        cout << std::setw(10) << Time / (k * n);

        readarray(input_file, array, n);
        for (unsigned int j = 0; j < n; ++j)
            m = array[m];
        asm("rdtsc\n":"=a"(start0.t32.th), "=d"(start0.t32.tl));
        bypass(array, n, k);
        asm("rdtsc\n":"=a"(end0.t32.th), "=d"(end0.t32.tl));
        Time = end0.t64 - start0.t64;
        cout << std::setw(10) << Time / (k * n);

        delete[] array;
        cout << endl;
    }
    return 0;
}
```

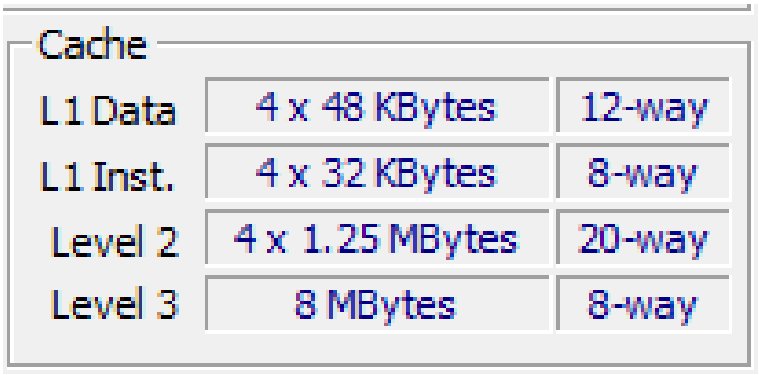
7. Оценка размера кэша

По графику можно заметить, что прирост тактов в рандомном обходе начинается с 48kB, что явно говорит о том, что размер L1 кэша равен 48kB.

Следующий подобное возрастание между 1 и 2 МБ, так что размер L2 кэша находится в этом диапазоне

В следующий раз более сильный прирост начинается после 8mB, что очень похоже на размер L3 кэша.

Данные, полученные с помощью CPU-Z, подтверждают данные предположения.



Данные верны для процессора Intel Core i5-11300h. Расчёты проводились при тактовой частоте 3.5GHz.

Стоит отметить, что не было замечено существенных различий между затратами на получение значения элемента массива при прямом и обратном обходах. Оба эти способа дали стабильные показатели тактов, которые изменились только при переходе на следующий уровень кэш, что было обеспечено правильной предвыборкой данных. Явно видна особенность быстродействия при обходе массива в случайном порядке: в этом случае количество тактов процессора, затраченное на получение элемента массива, значительно растёт с увеличением размера массива, что связано с тем, что кэш-контроллер не может корректно выполнить предвыборку данных для обхода массива в случайном порядке.

8. Заключение

В ходе данной работы я смог установить размеры кэшей собственного процессора и выяснить, насколько существенен прирост времени обращения при "переходе" с одного кэша на другой.