ЦЕЛЬ

1. Исследование зависимости времени доступа к данным в памяти от их объема.

2. Исследование зависимости времени доступа к данным в памяти от порядка их обхода.

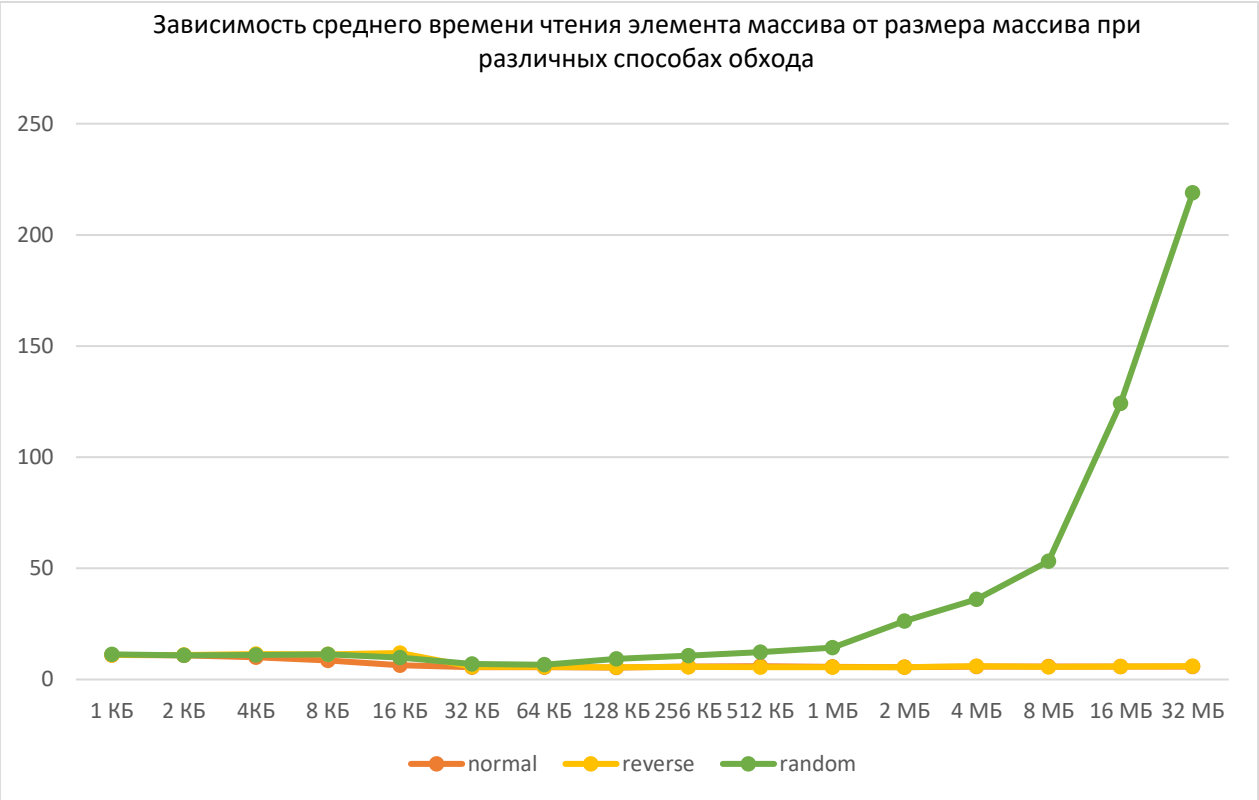
ОПИСАНИЕ РАБОТЫ

Для наблюдения за влиянием кэш-памяти на время обработки массивов, создадим три массива. Первый обходится по возрастанию адресов, второй по убыванию адресов, третий в рандомизированном порядке.

Составим график времени обхода в тактах процессора каждого массива:

объем памяти	normal	reverse	random
1 КБ	11.0382	11.0512	11.2997
2 КБ	10.8871	11.0501	10.8088
4КБ	10.059	11.4003	11.0512
8 КБ	8.62043	11.2289	11.3193
16 КБ	6.3667	11.9213	9.86174
32 КБ	5.49912	5.62752	6.94294
64 КБ	5.56721	5.67498	6.62908
128 КБ	5.34574	5.47848	9.28927
256 КБ	5.85187	5.61514	10.7205
512 КБ	6.0337	5.56102	12.2895
1 МБ	5.73015	5.5972	14.2993
2 МБ	5.53414	5.58183	26.2618
4 МБ	5.80835	5.91221	36.0839
8 МБ	5.75861	5.72551	53.2524
16 МБ	5.83637	5.82793	124.203
32 МБ	5.8509	5.89159	218.892

Построим график



Видим, что время обхода массива в порядке возрастания или убывания адресов почти одинаковая, однако при рандомном обходе время значительно растет с увеличением объема используемой памяти.

Это происходит потому, что когда мы хотим обратиться к элементу, который является соседним элементу, который мы уже рассмотрели, он с большой вероятностью находится в той же кэш-линии, а значит процессору не приходится обращаться в память и доставать из нее нужный элемент, ведь он уже находится к кэше. Однако для рандомного обхода невозможно точно предсказать к какому элементу мы обратимся следующим, и если этот элемент находится в другой кэш-линии (что при рандомном обходе более вероятно) придется обращаться в память намного чаще, что сильно замедлит время действия программы.

Размер кэша:

Кэш L1:	864 КБ
Кэш L2:	9,5 МБ
Кэш L3:	24,0 МБ

Видим, как до окончания первого уровня кэша время обхода почти не меняется, только для рандомного чуть возрастает, однако на втором уровне

заметны большие скачки, при переходе к третьему уровню время вовсе увеличивается больше чем в два раза.

ЗАКЛЮЧЕНИЕ

На практике удалось понять как кэш-память влияет на время обхода массива в зависимости от его объема и типа обхода. Обход по порядку занял существенно меньше времени, чем обход в рандомном порядке на большом объеме памяти.

Приложение 1. Листинг программы

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

void swap (int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void randomize (int* arr, int n) {
    srand (time(NULL));

    for (int i = n - 1; i > 0; i--) {
        int j = rand() % (i + 1);

        swap(&arr[i], &arr[j]);
    }
}

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

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

void create_randomized_array(int* array, int N) {
    create_array(array, N);
    randomize(array, N);
}

int main() {
    double total_time_normal = 0;
    double total_time_reversed = 0;
    double total_time_randomized = 0;


    int K = 128;

    int N_min = 256;
    int N_max = 8500000;
    for(int N = N_min; N < N_max; N *= 2) {
        int* array = (int*)calloc(sizeof(int), N);
        create_array(array, N);
        for (int k = 0, i = 0; i < N; i++) {
            k = array[k];
        }

        unsigned volatile long long start = __builtin_ia32_rdtsc();
        for (int k = 0, i = 0; i < N*K; i++) {
            k = array[k];
        }
```

```
        unsigned volatile long long finish = __builtin_ia32_rdtsc();

        total_time_normal = (finish - start) * 1.0 / (N * 1.0 * K);


        create_reversed_array(array, N);
        for (int k = 0, i = 0; i < N; i++) {
            k = array[k];
        }
        start = __builtin_ia32_rdtsc();
        for (int k = 0, i = 0; i < N*K; i++) {
            k = array[k];
        }
        finish = __builtin_ia32_rdtsc();
        total_time_reversed = (finish - start) * 1.0 / (N * 1.0 * K);

        create_randomized_array(array, N);
        for (int k = 0, i = 0; i < N; i++) {
            k = array[k];
        }
        start = __builtin_ia32_rdtsc();
        for (int k = 0, i = 0; i < N*K; i++) {
            k = array[k];
        }
        finish = __builtin_ia32_rdtsc();
        total_time_randomized = (finish - start) * 1.0 / (N * 1.0 * K);
        free(array);

        cout << (double) total_time_normal << endl;
        cout << (double) total_time_reversed << endl;
        cout << (double) total_time_randomized << endl;
        cout << endl;
    }
    return 0;
}
```