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 Бородина Артёма Максимовича 2 курса, 19205 группы

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н, доцент
А.Ю. Власенко

Новосибирск 2020

СОДЕРЖАНИЕ

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.

ЗАДАНИЕ

1. Написать программу, многократно выполняющую обход массива заданного размера тремя способами.
2. Для каждого размера массива и способа обхода измерить среднее время доступа к одному элементу (в тактах процессора). Построить графики зависимости среднего времени доступа от размера массива.
3. На основе анализа полученных графиков:
4. Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

1. Было изучено структура кэш-памяти процессора и их иерархия.

2. Была написана программа (Приложение 1), совершающая обход массива тремя способами: прямым, обратным и случайным. Также для каждого размера массива и способа было замерено среднее время доступа к элементу в тактах процессора.

3. По полученным данным был сделан график (Приложение 2) с шагом в степень двойки.

4. По графику можно увидеть, что время доступа к элементу при прямом и обратном способах почти одинаково и не превышает 6 тактов. Это связано с тем, что обход массива происходит предсказуемо для процессора и в кэш-память загружается нужная часть массива.

5. Время доступа к элементу случайным обходом резко возрастает после того, как размер массива становится близок к 2МБ, что равно кэш-памяти 2го уровня. Это связано с тем, что в кэш-память 2го уровня перестает помещаться полностью весь массив, а случайный обход мешает тому, чтобы в кэш-памяти находилась нужная часть массива.

6. Также заметен скачок после 16КБ потому, что размер массива стал превышать размеры кэша 1го уровня.

ЗАКЛЮЧЕНИЕ

Были получены знания о работе кэш-памяти, изучена взаимосвязь скорости доступа к элементу массива в зависимости от его размера и обхода.

Было доказано, что при размерах массива меньших размера кэш-памяти времена обхода массива разными способами приблизительно равны.

Приложение 1. *Код программы*

```
#include <iostream>
#include <algorithm>
#include <random>
#include <ctime>
#include <fstream>
#include <iomanip>
#include <intrin.h>

#define TIME_MEASURE_COUNT 10
#define REPEAT 2
#define MEASURE_AMOUNT 10

using namespace std;

double MeasureTime(const int *array, int size) {
    unsigned long long minDuration = ULONG_LONG_MAX;
    int k;
    for (int i = 0; i < TIME_MEASURE_COUNT; ++i) {
        unsigned long long start = __rdtsc();
        k = 0;
        for (int j = 0; j < size * REPEAT; ++j) {
            k = array[k];
        }
        unsigned long long end = __rdtsc();
        unsigned long long tmpDuration = end - start;
        if (tmpDuration < minDuration) {
            minDuration = tmpDuration;
        }
    }
    cerr << k << endl;
    return minDuration / double(size * REPEAT);
}

int main() {
    int n = 256;
    int mMax = 1024 * 16;

    ofstream output("file8.csv");
    if (!output) {
        cout << "Cannot open output file";
        return 0;
    }
    output << ";direct;reverse;random" << endl;

    for (int m = 1, iter = 1; m < mMax; m += 16, ++iter) {
        int size = n * m;
        long elapsedTime = -clock();
        int directTraversal[size];
        int reverseTraversal[size];
        int randomTraversal[size];
        int shuffledArray[size];

        for (int i = 0; i < size; ++i) {
            shuffledArray[i] = i;
            directTraversal[i] = i + 1;
            reverseTraversal[size - i - 1] = size - i - 2;
        }
        directTraversal[size - 1] = 0;
        reverseTraversal[0] = size - 1;
        shuffle(shuffledArray, shuffledArray + size, default_random_engine(time(nullptr)));

        for (int i = 0; i < size - 1; ++i) {
            randomTraversal[shuffledArray[i]] = shuffledArray[i + 1];
        }
        randomTraversal[shuffledArray[size - 1]] = shuffledArray[0];

        double directDuration = 0;
        double reverseDuration = 0;
        double randomDuration = 0;

        for (int i = 0; i < MEASURE_AMOUNT; ++i) {
            directDuration += MeasureTime(directTraversal, size);
            reverseDuration += MeasureTime(reverseTraversal, size);
            randomDuration += MeasureTime(shuffledArray, size);
        }

        cout << setprecision(numeric_limits<double>::digits10 + 1);
        cout << endl << "iter: " << iter << " memoryPerArray: " << size / 256 << "KB" << endl
             << "n: " << n << " m: " << m << endl
             << "directDuration: " << directDuration / MEASURE_AMOUNT << endl
             << "reverseDuration: " << reverseDuration / MEASURE_AMOUNT << endl
             << "randomDuration: " << randomDuration / MEASURE_AMOUNT << endl;

        output << size / 256 << ";"
               << directDuration / MEASURE_AMOUNT << ";"
               << reverseDuration / MEASURE_AMOUNT << ";"
               << randomDuration / MEASURE_AMOUNT << endl;
        elapsedTime += clock();
        cout << "elapsedTime: " << elapsedTime / double(CLOCKS_PER_SEC) << endl;
    } return 0;
}
```

Приложение 2. *Замеры среднего времени доступа к элементу*