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, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

- *Исследование зависимости времени доступа к данным в памяти от их объема;*
- *Исследование зависимости времени доступа к данным в памяти от порядка их обхода;*

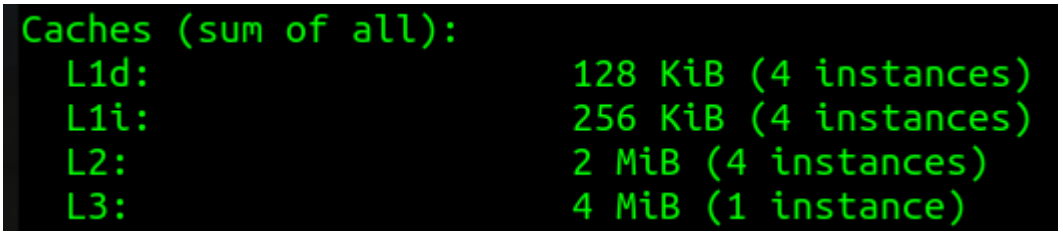
ЗАДАНИЕ

- *Написать программу, многократно выполняющую обход массива заданного размера тремя способами;*
- *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.*
- *На основе анализа полученных графиков:*
    - *Определить размеры кэш-памяти различных уровней, обосновать ответ, сопоставить результат с известными реальными значениями;*
    - *Определить размеры массива, при которых время доступа к элементу массива при случайном обходе больше, чем при прямом или обратном; объяснить причины этой разницы во временах.*
- *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
    - *Титульный лист;*
    - *Цель лабораторной работы;*
    - *Описание способа заполнения массива тремя способами;*
    - *Графики зависимости среднего времени доступа к одному элементу от размера массива и способов обхода;*
    - *Полный компилируемый листинг реализованной программы и команду для ее компиляции;*
    - *Вывод по результатам лабораторной работы.*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Написал программу, многократно выполняющую прямой, обратный и произвольный обходы массива заданного размера и измеряющую среднее время доступа к одному элементу (в тактах процессора);*
2. *Запустил команду **lscpu**, чтобы узнать реальные значения размера кэш-памяти разного уровня;*

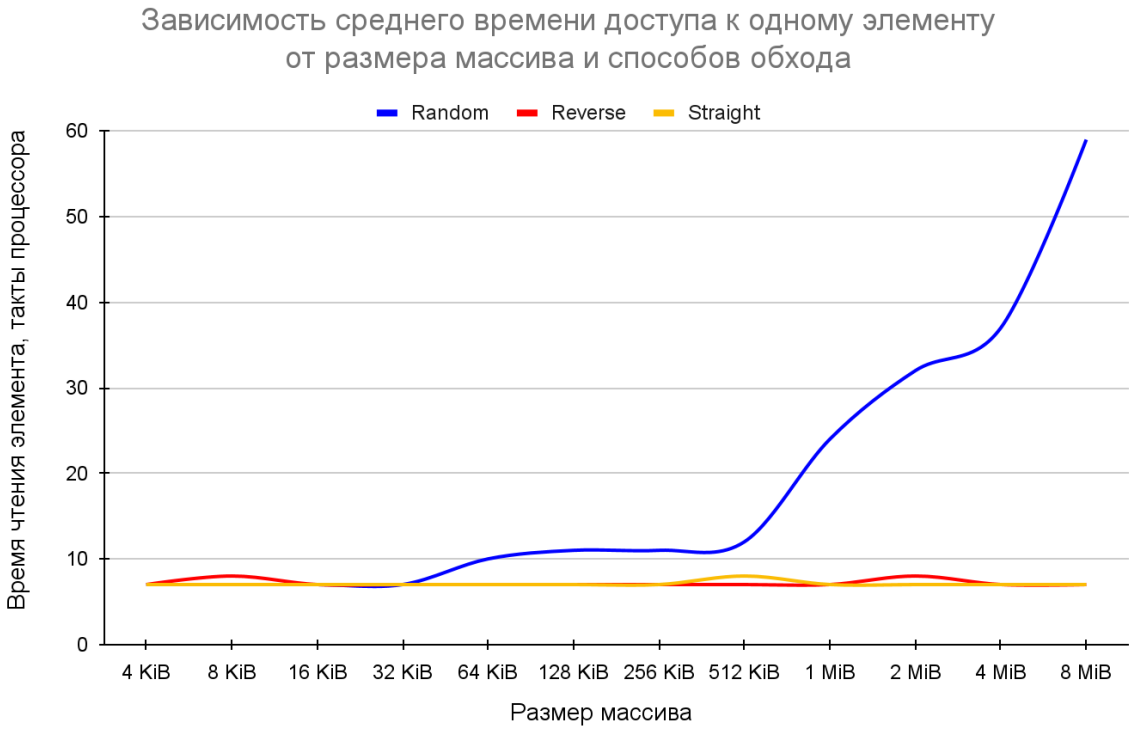


3. *Построил графики зависимости среднего времени доступа от размера массива для каждого способа обхода (см. Результаты измерения);*

Команды для компиляции

```
➜ lab8 ✗ cmake -DCMAKE_BUILD_TYPE=Debug -G Ninja -B
cmake-build-debug
➜ lab8 ✗ cmake --build cmake-build-debug --target lab8
➜ lab8 ✗ cmake-build-debug/lab8
```

Результаты измерения


Зависимость среднего времени доступа к одному элементу
от размера массива и способов обхода
Random
Reverse
Straight
Время чтения элемента, такты процессора
60
50
40
30
20
10
0
4 KiB
8 KiB
16 KiB
32 KiB
64 KiB
128 KiB
256 KiB
512 KiB
1 MiB
2 MiB
4 MiB
8 MiB
Размер массива

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

- *Исследовалась зависимость времени доступа к данным в памяти от их объема;*
- *Исследовалась зависимость времени доступа к данным в памяти от порядка их обхода;*

По результатам проведённых исследований можно сделать следующие выводы:

- *По приросту времени доступа к данным в памяти можно определить размеры кэш-памяти разных уровней (L1 - 32 КБ, L2 - 512 КБ, L3 - 4 МБ);*
- *При прямом и обратном обходе работает аппаратная предвыборка данных, поэтому время доступа к данным в памяти не зависит от размера массива;*
- *При случайном обходе не работает аппаратная предвыборка данных, поэтому время доступа к данным в памяти нелинейно зависит от размера массива (с увеличением размера массива время доступа к данным в памяти ступенчато возрастает);*

ПРИЛОЖЕНИЕ (ЛИСТИНГ ПРОГРАММЫ)

src/main.cpp

```cpp
#include <cstdint>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <set>

#define MIN_SIZE (1 * 1024)
#define MAX_SIZE (2 * 1024 * 1024)

using Time = uint64_t;

void Clock(Time & time);
void CreateStraight(uint32_t * array, uint32_t size);
void CreateReverse(uint32_t * array, uint32_t size);
void CreateRandom(uint32_t * array, uint32_t size);
uint64_t CalcTime(const uint32_t * array, uint32_t size);
void Bypass(const uint32_t * array, uint32_t size);

int main()
{
    std::cout << std::setw(10) << std::left << "Size"
              << std::setw(10) << std::left << "Random"
              << std::setw(10) << std::left << "Reverse"
              << std::setw(10) << std::left << "Straight"
              << std::endl;

    for (uint32_t size = MIN_SIZE; size <= MAX_SIZE; size *= 2)
    {
        std::cout << std::setw(10) << std::left
                  << size * sizeof(uint32_t);

        auto * array = new uint32_t[size];

        CreateRandom(array, size);
        std::cout << std::setw(10) << std::left
                  << CalcTime(array, size);

        CreateReverse(array, size);
        std::cout << std::setw(10) << std::left
                  << CalcTime(array, size);

        CreateStraight(array, size);
        std::cout << std::setw(10) << std::left
                  << CalcTime(array, size) << std::endl;

        delete [] array;
    }

    return EXIT_SUCCESS;
}
```

```
void Clock(Time & time)
{
    asm("rdtsc\n":"=a"(time));
}

void CreateStraight(uint32_t * array, uint32_t size)
{
    array[size - 1] = 0;
    for (uint32_t i = 0; i < size - 1; ++i)
        array[i] = i + 1;
}

void CreateReverse(uint32_t * array, uint32_t size)
{
    array[0] = size - 1;
    for (uint32_t i = 1; i < size; ++i)
        array[i] = i - 1;
}

void CreateRandom(uint32_t * array, uint32_t size)
{
    std::set<uint32_t> indexes;
    for (uint32_t i = 0; i < size - 1; ++i)
        indexes.insert(i + 1);

    srandom(time(nullptr));
    uint32_t ind = 0;

    while (!indexes.empty())
    {
        uint32_t indexes_ind = random() % indexes.size();
        auto next_ind_iter = std::next(indexes.begin(), indexes_ind);
        array[ind] = *next_ind_iter;
        ind = *next_ind_iter;
        indexes.erase(next_ind_iter);
    }

    array[ind] = 0;
}

void Bypass(const uint32_t * array, uint32_t size)
{
    for (uint32_t i = 0, k = 0; i < size; ++i)
        k = array[k];
}

uint64_t CalcTime(const uint32_t * array, uint32_t size)
{
    uint64_t min = UINT64_MAX;
    Time start{}, end{};

    for (int i = 0; i < 1; ++i)
    {
        Clock(start);
        Bypass(array, size);
        Clock(end);
```

```
            uint64_t res = (end - start) / size;
            min = (min < res) ? min : res;
        }

        return min;
    }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.3)
project(lab8 CXX)
add_executable(lab8 main.cpp)
```