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, 23209 группы
Инокова Семёна Шухратовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А. Ю. Кудинов

Новосибирск 2024

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:

- 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.

ОПИСАНИЕ РАБОТЫ

1. Написана программа на языке Си, которая многократно выполняет обход массива тремя способами: последовательно в сторону увеличения адресов, последовательно в сторону уменьшения адресов и в случайном порядке.

2. Размер заданного массива изменяется с 1 KiB до 32 MB.

3. Полученные результаты после каждого обхода сохраняются в файлы.

3. Построил график, согласно результатам работы программы, с помощью графика сделал вывод, что размер кэш-памяти первого уровня равен 38 KiB, второго — 282 KiB, и третьего — 8 MB.

4. С помощью команды lscpu получил реальные данные о размерах уровней кэш-памяти.

5. Проанализировал и сравнил результаты.

ЗАКЛЮЧЕНИЕ

В ходе данной работы я исследовал зависимости времени доступа к данным в зависимости от их объема и порядка их обхода. При увеличении размера массива и выходе из уровня кэш-памяти, увеличивается количество кэш-промахов и затрачиваемое время на обход массива резко увеличивается.
С помощью графика удалось выяснить приближенные размеры уровней кэш-памяти.

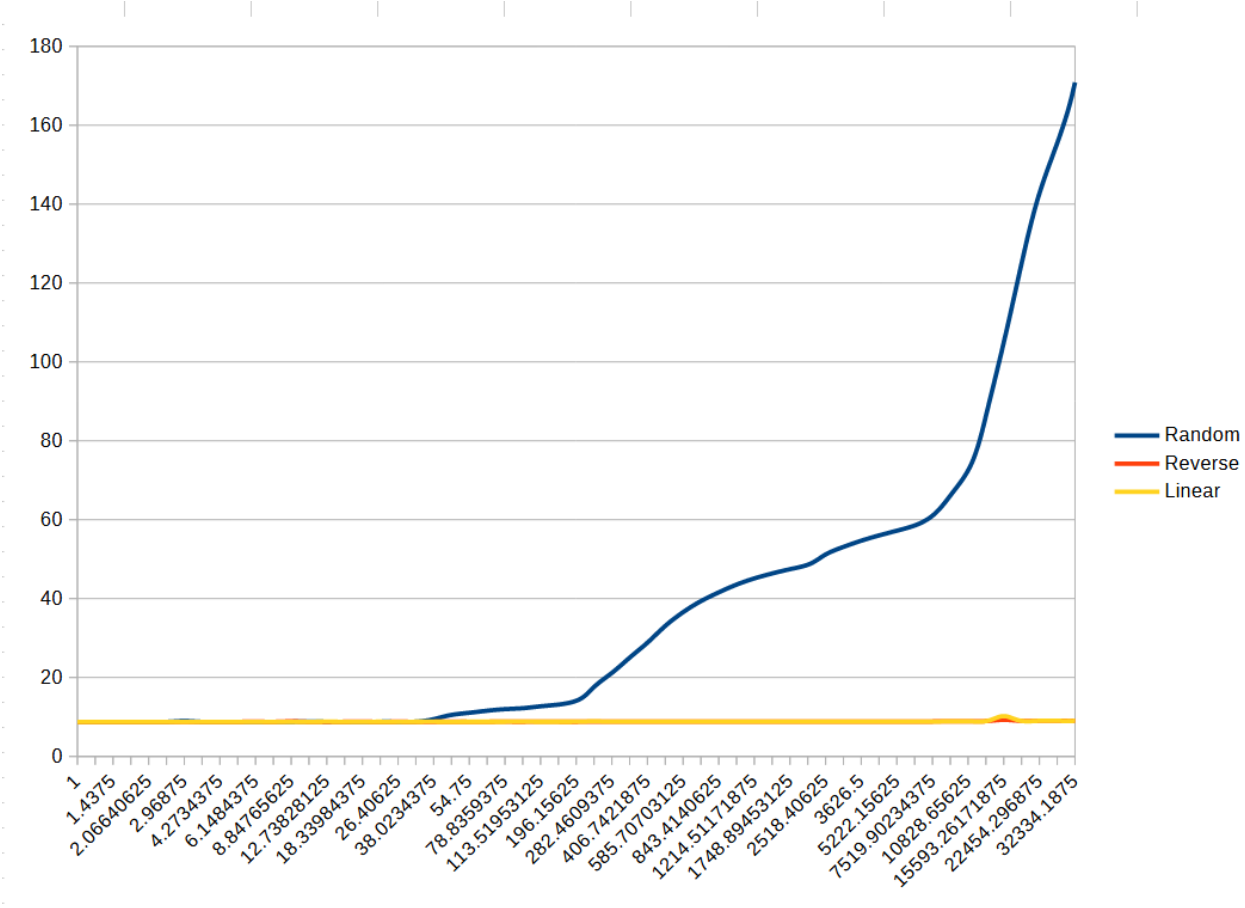
Приложение 1. *Код программы для работы*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

#define NMIN 256
#define NMAX (32 * 1024 * 1024 / sizeof(int))
#define REPEATS 100

#if defined(_MSC_VER)
#include <intrin.h>
uint64_t rdtsc() {
    return __rdtsc();
}
#elif defined(__GNUC__) || defined(__clang__)
uint64_t rdtsc() {
    unsigned int lo, hi;
    __asm__ __volatile__("rdtsc" : "=a"(lo), "=d"(hi));
    return ((uint64_t)hi << 32) | lo;
}
#else
#error "RDTSC not supported on this platform"
#endif

void multMatrix()
{
    const size_t size = 2048;
    float *A = malloc(size * size * sizeof(float));
    float *B = malloc(size * size * sizeof(float));
    float *C = calloc(size * size, sizeof(float));
    for (size_t i = 0; i < size; i++)
        for (size_t k = 0; k < size; k++)
        {
        float a = A[i * size + k];
        for (size_t j = 0; j < size; j++)
            C[i * size + j] += a * B[k * size + j];
        }
    printf("%f %f %f\n", C[0], C[size * size - 1], C[size + 1]);
    free(A);
    free(B);
    free(C);
}

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

```
void fill_reverse(int *array, int size) {
    for (int i = 0; i < size; i++) {
        array[i] = (i - 1 + size) % size;
    }
}

void fill_random(int *array, int size) {
    for (int i = 0; i < size; i++) {
        array[i] = i;
    }
    for (int i = 0; i < size; i++) {
        int j = rand() % size;
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}

uint64_t measure_cycles(int *array, int size) {
    volatile int k = 0;
    uint64_t start = rdtsc();
    for (int r = 0; r < REPEATS; r++) {
        for (int i = 0; i < size; i++) {
            k = array[k];
        }
    }
    uint64_t end = rdtsc();
    return (end - start);
}

int main() {
    multMatrix();

    FILE *file_direct = fopen("direct_cycles.csv", "w");
    FILE *file_reverse = fopen("reverse_cycles.csv", "w");
    FILE *file_random = fopen("random_cycles.csv", "w");

    if (!file_direct || !file_reverse || !file_random) {
        fprintf(stderr, "Error opening output files\n");
        exit(EXIT_FAILURE);
    }

    fprintf(file_direct, "Size (elements), Cycles per element\n");
    fprintf(file_reverse, "Size (elements), Cycles per element\n");
    fprintf(file_random, "Size (elements), Cycles per element\n");

    for (int size = NMIN; size <= NMAX; size *= 1.2) {
        int *array = (int *)malloc(size * sizeof(int));
        if (!array) {
            fprintf(stderr, "Memory allocation failed for size %d\n", size);
            exit(EXIT_FAILURE);
        }
```

```
        // Прямой обход
        fill_sequential(array, size);
        uint64_t cycles_direct = measure_cycles(array, size);
        double cycles_per_element_direct = (double)cycles_direct / (size * REPEATS);
        fprintf(file_direct, "%d, %.3f\n", size, cycles_per_element_direct);

        // Обратный обход
        fill_reverse(array, size);
        uint64_t cycles_reverse = measure_cycles(array, size);
        double cycles_per_element_reverse = (double)cycles_reverse / (size * REPEATS);
        fprintf(file_reverse, "%d, %.3f\n", size, cycles_per_element_reverse);

        // Случайный обход
        fill_random(array, size);
        uint64_t cycles_random = measure_cycles(array, size);
        double cycles_per_element_random = (double)cycles_random / (size * REPEATS);
        fprintf(file_random, "%d, %.3f\n", size, cycles_per_element_random);

        free(array);
    }

    fclose(file_direct);
    fclose(file_reverse);
    fclose(file_random);

    return 0;
}
```

Приложение 2. *График зависимости времени работы программы от размера массива (KiB)*



Приложение 3. *Размеры уровней кэш-памяти согласно lscpu*