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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Влияние кэш-памяти на время обработки массивов»

студента 2 курса, группы 22206

***Тропина Никиты Васильевича***

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент, к.т.н.

А.Ю.Власенко

Новосибирск 2023

# СОДЕРЖАНИЕ

[ЦЕЛИ 2](#_Toc152805665)

[ЗАДАНИЕ 2](#_Toc152805666)

[ОПИСАНИЕ РАБОТЫ 3](#_Toc152805667)

[ЗАКЛЮЧЕНИЕ 6](#_Toc152805668)

[Приложение 1. Программа, измеряющая среднее время обращения к элементам массивов различных размеров. 7](#_Toc152805669)

# ЦЕЛИ

1. Оценить размеры кэш-памяти различных уровней.
2. Определить время доступа к кэш-памяти различных уровней.

# ЗАДАНИЕ

1. Написать программу, многократно выполняющую обход массива заданного размера тремя способами (прямой, обратный и случайный).
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. Каждый последующий размер массива отличается от предыдущего **не более, чем в 1,2 раза**.
3. Определить размеры кэш-памяти точным образом (на основе документации по процессору используемой машины; утилите, отражающей характеристики процессора; системному файлу;…)
4. На основе полученных графиков.

* Оценить 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.

1. Составить отчет по лабораторной работе.

# ОПИСАНИЕ РАБОТЫ

Для точного измерения времени необходимо перед измерением нагрузить процессор, чтобы процессор поднял тактовую частоту и установил её на фиксированном уровне. Иначе в начале замеряемого промежутка времени значения на графике будут выше.

Также необходимо делать «прогрев кэша» для вытеснения ненужных данных. Это достигается прогоном цикла без замера времени.

Так как базовой единицей измерения времени работы процессора являются такты, то и производить вычисления времени будем с помощью таймера rdtsc, возвращающего именно количество тактов.

К замеру подлежит только цикл, проходящий по всем элементам массива. Его размер на каждом новом прогоне умножается на 1.2 для получения плавного графика зависимости времени доступа к памяти от размера массива.

Учитывая вышеизложенные замечания, была написана программа, представленная в Приложении 1. А также получены результаты времени прямого, обратного и случайного обходов массива.

Для случайного обхода использовался алгоритм:

1. Случайно выбирается значение 1 – (N – 1), где N – размер массива.
2. Выбранное число является позицией, на которую будет записан 0.
3. Далее выбираются оставшиеся числа из указанного диапазона без повторений, но на каждое новое место записывается номер предыдущего выбранного элемента.
4. После выбора всех чисел из отрезка 1 – (N - 1), на нулевое место массива записывается номер последнего выбранного элемента.

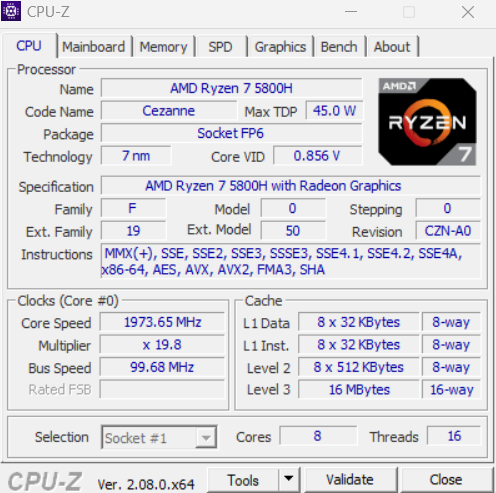
Данный метод гарантирует обход всех элементов массива за N шагов в случайном порядке при обращении к значения следующим образом:

Все полученные значения представлены на диаграмме 1.

Диаграмма 1.

Как видно из диаграммы, при заметном увеличении времени доступа, размер массива начинает располагать в кэш-памяти следующего уровня. На испытуемой машине имеет 3 уровня кэш-памяти процессора. Размерность L1 = 256 КБ, L2 = 1 МБ и L3 = 16 МБ.

Для получения точных данных использовалась программа CPU-Z.



Сравнивая полученные результаты с настоящими, сошлись два из 3 размера кэш-памяти: L1, L3. Размерность L2, согласно CPU-Z, равна 4 МБ, а не 1, как было определено по графику.

# ЗАКЛЮЧЕНИЕ

Проведя замеры времени на время обращение к элементу массива при разных размерах, получили данные, сходящиеся с фактическими: чем больше размер массива, тем дольше идет обращение к памяти при случайном обходе и всегда постоянное при равномерном обходе. Это связано с тем, что процессор различает простые обходы памяти и заранее подгружает нужные данные в L1. При случайном же, постоянно происходят кэш-промахи, из-за которых данные приходится искать в памяти более высокого уровня, что существенно сказывается на времени доступа к ней.

# Приложение 1. Программа, измеряющая среднее время обращения к элементам массивов различных размеров.

#pragma optimize("O1", "on")

#include <iostream>

#include <intrin.h>

#include <math.h>

#include <unordered\_map>

#define RAND\_MAX 8388608

static unsigned long int next = 1;

int warmUp(int seconds)

{

int size = 1024;

std::vector<int> a(size);

std::vector<int> b(size);

for (int l = 0; l < seconds; l++) {

for (int k = 0; k < 64; k++) {

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

a[j] = i \* j + 1;

b[j] = a[i] \* j;

a[j] \*= b[j] / 2;

}

}

}

}

if (a[size / 2] == b[size / 2]) {

return 1;

}

else {

return 0;

}

}

void fillArrayStraight(unsigned int array[], unsigned int size)

{

for (unsigned int i = 0; i < size - 1; i++) {

array[i] = i + 1;

}

array[size - 1] = 0;

}

void fillArrayReverse(unsigned int array[], unsigned int size)

{

for (unsigned int i = 0; i < size - 1; i++) {

array[size - 1 - i] = size - 2 - i;

}

array[0] = size - 1;

}

int rand(void) {

next = next \* 1103515245 + 12345;

return (unsigned int)(next) % (RAND\_MAX + 1);

}

void fillArrayRandom(unsigned int array[], unsigned int size) {

std::unordered\_map<unsigned int, unsigned int> myMap;

unsigned int j = rand() % (size - 1) + 1;

unsigned int previouseIndex = 0;

for (unsigned int i = 0; i < size - 1; i++) {

while (myMap.find(j) != myMap.end()) {

j = rand() % (size - 1) + 1;

}

myMap.emplace(j, 1);

array[j] = previouseIndex;

previouseIndex = j;

}

array[0] = previouseIndex;

}

unsigned int arrayTraversal(unsigned int array[], unsigned int size, unsigned int nRepetitions)

{

unsigned \_\_int64 tactsResult = UINT64\_MAX;

for (unsigned int i = 0; i < nRepetitions; i++) {

unsigned \_\_int64 curTactsResult = 0;

unsigned int index = 0;

unsigned \_\_int64 tacts = \_\_rdtsc();

for (unsigned int j = 0; j < size; j++) {

index = array[index];

}

curTactsResult += \_\_rdtsc() - tacts;

if (curTactsResult < tactsResult) {

tactsResult = curTactsResult;

}

curTactsResult = 0;

}

return tactsResult / size;

}

unsigned int\* arrayTraversalWithSizeChanging(unsigned int array[], unsigned int size, unsigned int lastSize, float step, int nRepetitions, int mode)

{

int nSteps = log(lastSize / size) / log(1.2) + 1;

unsigned int\* tacts = new unsigned int[nSteps + 1] {};

tacts[0] = nSteps + 1;

for (int i = 0; i < nSteps; i++) {

array = new unsigned int[size] {};

if (mode == 0) {

fillArrayStraight(array, size);

}

else if (mode == 1) {

fillArrayReverse(array, size);

}

else {

fillArrayRandom(array, size);

}

// warm up

tacts[i + 1] = arrayTraversal(array, size, 1);

// real work

tacts[i + 1] = arrayTraversal(array, size, nRepetitions);

std::cout << "size: " << size << ", tacts: " << tacts[i + 1] << std::endl;

size \*= step;

}

return tacts;

}

int main()

{

unsigned int size = 256;

int nRepetitions = 10;

float step = 1.2;

int mode = 1;

unsigned int \*array = new unsigned int[size];

std::cout << warmUp(4) << std::endl;

unsigned int \*tacts = arrayTraversalWithSizeChanging(array, size, 66322432, step, nRepetitions, mode);

std::cout << arrayTraversal(array, size, nRepetitions) << std::endl;

for (int i = 0; i < tacts[0]; i++) {

std::cout << tacts[i] << " ";

}

}