###### 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, группы 19201

Хаецкой Дарьи Владимировны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2020

1. **Цель**

Изучить принцип работы работа кэш-память процессора и на примере обхода элементов массива в разном порядке выяснить размер разных уровней кэш-памяти на конкретном ПК.

1. **Задание**

На основе разных способов обхода циклического массива: прямого, обратного и случайного, сделать среднюю оценку количества тактов для обращения к одному элементу массива. На основе полученных данных выяснить приблизительный размер кэша процессора.

1. **Описание обходов**

Используются следующие обходы массива:

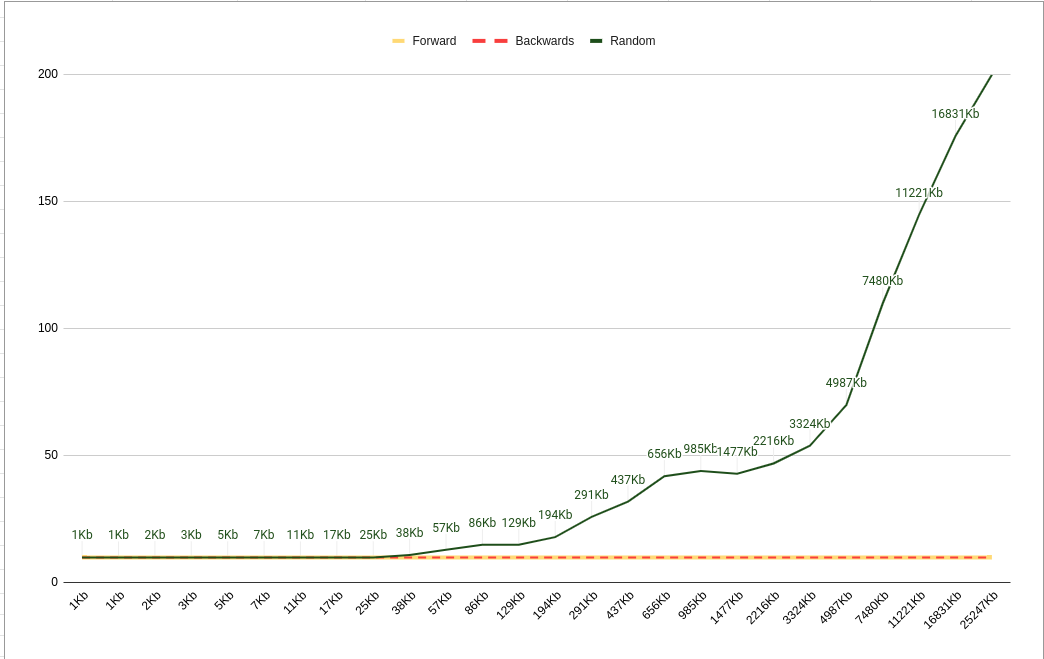
* Прямой;
* Обратный;
* Случайный.

**Прямой обход** подразумевает, что значением ячейки массива с индексом i будет i+1, т.е. индекс следующей соседней ячейки массива, в случае последнего элемента следующим элементом будет самый первый. Таким образом мы получаем следующий индекс и так обходим массив несколько раз.

**Обратный обход** – прямой обход, но с условием, что теперь обход идёт с конца массива линейным порядком.

**Случайный обход** подразумевает, что значением ячейки массива с индексом i будет некоторый индекс, который невозможно предугадать, но гарантируется, что индекс не выйдет за пределы массива. Тем самым, ходя по разным индексам, мы полностью обходим массив.

1. **График роста тактов обращения**



1. **Листинг программы**

#include <iostream>

#include <set>

#include <cstdlib>

#include <ctime>

#include <fstream>

#include <limits.h>

using namespace std;

void goForward(unsigned int \*array, unsigned int n){

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

array[i] = i + 1;

}

array[n - 1] = 0;

}

void goBackwards(unsigned int \*array, unsigned int n){

for(unsigned int i = n - 1; i > 0; --i)

array[i] = i - 1;

array[0] = n - 1;

}

void goRandom(unsigned int \*array, const unsigned int arraySize) {

auto \*indexes = new unsigned int[arraySize];

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

indexes[i] = i;

}

srand(time(NULL));

for (long long i = arraySize - 1; i >= 0; --i) {

unsigned int j = 0;

if (i > 0) {

j = rand() % i;

}

swap(indexes[i], indexes[j]);

}

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

array[indexes[i]] = indexes[i + 1];

}

array[indexes[arraySize - 1]] = indexes[0];

delete[] indexes;

}

void run(const unsigned int \*array, unsigned int n, unsigned int k)

{

int m = 0;

for(unsigned int j = 0; j < n \* k; ++j)

m = array[m];

}

int main(){

ofstream log;

log.open("log.txt");

unsigned int minSize = 256;

unsigned int k = 10;

int maxSize = 8 \* 1024 \* 1024; //32 Mb

unsigned int currentSize = minSize;

unsigned int \*array;

for (; currentSize < maxSize; currentSize = (unsigned int)(currentSize \* 1.5)){

array = (unsigned int \*)malloc(currentSize \* sizeof(unsigned int));

goForward(array, currentSize);

unsigned long long minTime = LLONG\_MAX;

run(array, currentSize, 3);

for (int j = 0; j < 3; j++) {

unsigned long long start = \_\_rdtsc();

run(array, currentSize, k);

unsigned long long end = \_\_rdtsc();

unsigned long long Time = end - start;

if (Time < minTime) {

minTime = Time;

}

}

log << "Forward lookup. For " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

cout << "Forward lookup. For " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

free(array);

}

currentSize = minSize;

for (; currentSize < maxSize; currentSize = (unsigned int)(currentSize \* 1.5)){

array = (unsigned int \*)malloc(currentSize \* sizeof(unsigned int));

goBackwards(array, currentSize);

unsigned long long minTime = LLONG\_MAX;

run(array, currentSize, 3);

for (int j = 0; j < 3; j++) {

unsigned long long start = \_\_rdtsc();

run(array, currentSize, k);

unsigned long long end = \_\_rdtsc();

unsigned long long Time = end - start;

if (Time < minTime) {

minTime = Time;

}

}

log << "Backwards lookup for " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

cout << "Backwards lookup for " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

free(array);

}

currentSize = minSize;

for (; currentSize < maxSize; currentSize = (unsigned int)(currentSize \* 1.5)){

array = (unsigned int \*)malloc(currentSize \* sizeof(unsigned int));

goRandom(array, currentSize);

unsigned long long minTime = LLONG\_MAX;

run(array, currentSize, 3);

for (int j = 0; j < 3; j++) {

unsigned long long start = \_\_rdtsc();

run(array, currentSize, k);

unsigned long long end = \_\_rdtsc();

unsigned long long Time = end - start;

if (Time < minTime) {

minTime = Time;

}

}

log << "Random lookup for " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

cout << "Random lookup for " << currentSize \* 32 / 8 / 1024

<< "Kb, taken: " << minTime / (k \* currentSize) << endl;

free(array);

}

log.close();

return 0;

}

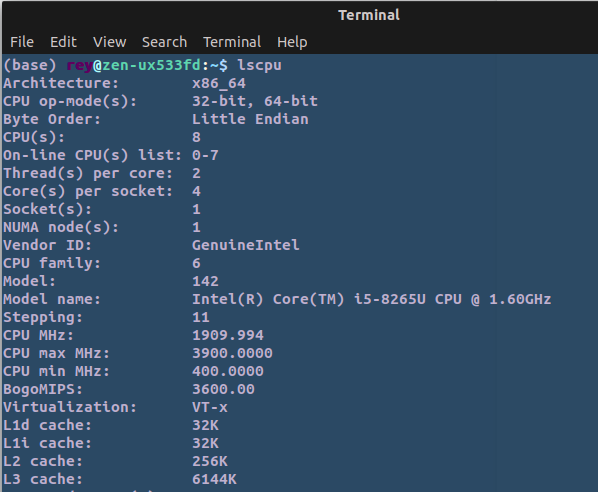
1. **Оценка размера кэша**

По графику можно заметить, что прирост тактов начинается с 32kB, что явно говорит о том, что размер L1 кэша равен 32kB.

Следующее возрастание между 194Kb и 294Kb, что соответствует размеру L2 кэша - 256Kb.

В следующий раз более сильный прирост начинается после 5000Kb, что очень похоже на размер L3 кэша.

Данные, полученные c помощью команды lscpu подтверждают данные предположения.



Вследствие предвыборки данных, прямой и обратный обходы не росли с увеличением размера массива. Зато видна особенность обхода массива в случайном порядке: в этом случае количество тактов процессора, затраченное на получение элемента массива, заметно растет с увеличением размера массива потому что что кэш-контроллер не может корректно выполнить предвыборку данных для обхода массива в случайном порядке.

1. **Заключение**

В ходе данной работы были установлены размеры кэшей процессора i5-8265U и выяснено, насколько существенен прирост времени обращения при "переходе" с одного кэша на другой.