###### 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 21209

Усольцева Антона Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

М.А.Мичуров

В.А.Перепёлкин

Новосибирск 2022

**СОДЕРЖАНИЕ**

Цель 3

Задачи 3

Описание работы 4

Заключение 7

# ЦЕЛИ

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.

# ЗАДАЧИ

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

Листинг 1

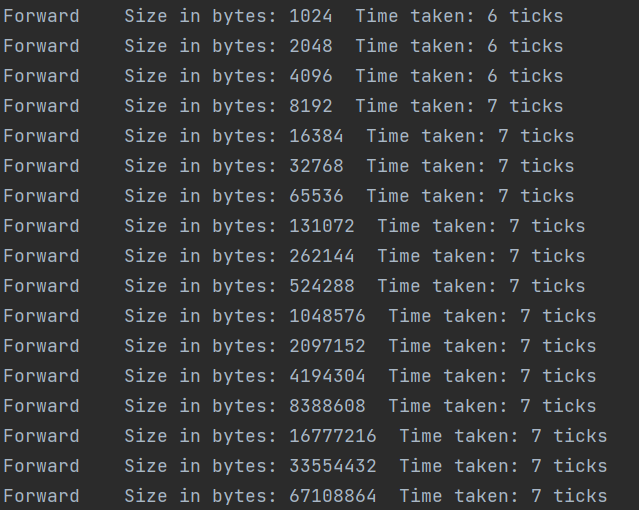
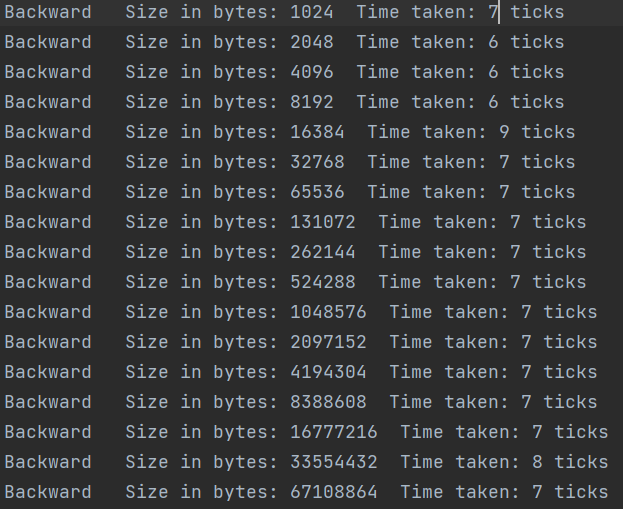
#include <iostream>  
#include <fstream>  
#include <random>  
#include <algorithm>  
#include <vector>  
  
#define N\_min 256  
#define N\_max 16777216  
#define K 10  
  
int \*fill\_in\_forward(size\_t size) {  
 auto arr = new int[size];  
 for (int i = 0; i < size - 1; i++)   
 arr[i] = i + 1;  
 arr[size - 1] = 0;  
 return arr;  
}  
  
int \*fill\_in\_backward(size\_t size) {  
 auto arr = new int[size];  
 for (int i = 1; i < size; i++)  
 arr[i] = i - 1;  
 arr[0] = static\_cast<int> (size - 1);  
 return arr;  
}  
  
int \*fill\_in\_random(int size) {  
 auto arr = new int[size];  
 std::vector<int> ind(size - 1);  
 for (int i = 0; i < size - 1; i++)  
 ind[i] = i + 1;

int cur = 0, prev = 0;  
 std::shuffle(ind.begin(),ind.end(),std::mt19937(std::random\_device()()));

for (int i = 0; i < size - 1 && !ind.empty(); i++) {  
 cur = ind[ind.size() - 1];  
 ind.pop\_back();  
 arr[prev] = cur;  
 prev = cur;  
 }  
 arr[cur] = 0;  
 return arr;  
}  
  
void cache\_warm(const int \*arr, size\_t size) {  
 for (int k = 0, i = 0; i < size; i++)  
 k = arr[k];  
}  
  
void bypass(const int \*arr, size\_t size, std::ofstream &fout) {  
 union ticks {  
 unsigned long long t64;  
 struct s32 {  
 long th, tl;  
 } t32;  
 } start{}, end{};  
  
 cache\_warm(arr, size);

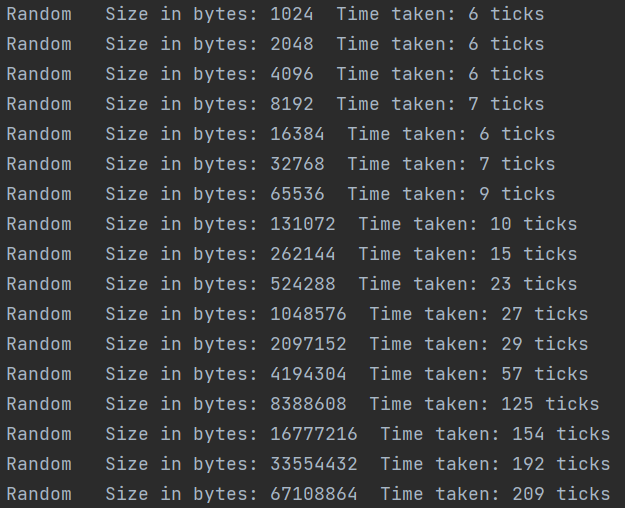
asm volatile ("rdtsc\n":"=a"(start.t32.th),"=d"(start.t32.tl)::"memory");  
  
 for (int k = 0, i = 0; i < size \* K; i++)  
 k = arr[k];  
  
 asm volatile ("rdtsc\n":"=a"(end.t32.th), "=d"(end.t32.tl)::"memory");  
 fout << "Count of int elements: " << size << " ";  
 fout << "Time taken: " << (end.t64 - start.t64)/(size\*K) << " ticks\n";  
 fout.flush();  
}  
  
int main() {  
 int \*test\_arr;  
 std::ofstream forw\_fout("forw\_res.txt");  
 std::ofstream backw\_fout("backw\_res.txt");  
 std::ofstream rand\_fout("rand\_res.txt");  
 if (!forw\_fout.is\_open()||!backw\_fout.is\_open()||!rand\_fout.is\_open())  
 return 1;  
  
 for (int s = N\_min; s <= N\_max; s \*= 2) {   
 test\_arr = fill\_in\_forward(s);  
 forw\_fout << "Forward ";  
 bypass(test\_arr, s, forw\_fout);  
 delete[] test\_arr;  
  
 test\_arr = fill\_in\_backward(s);  
 backw\_fout << "Backward ";  
 bypass(test\_arr, s, backw\_fout);  
 delete[] test\_arr;  
  
 test\_arr = fill\_in\_random(s);  
 rand\_fout << "Random ";  
 bypass(test\_arr, s, rand\_fout);  
 cache\_warm(test\_arr, s);  
 delete[] test\_arr;  
 }  
}

В основном цикле **for (int s = N\_min; s <= N\_max; s \*= 2)** использован нелинейный шаг для того, чтобы проверить несколько маленьких значений и при этом не проверять много больших.

 Рассмотрим выводимые данные и построим график зависимости среднего времени обращения от размера массива.

Прямой обход

Обратный обход



Обход элементов в случайном порядке.

Построим график:

Зависимость среднего времени чтения элемента массива от размера массива при различных способах обхода для процессора Intel Core i5 8300H

Из конфигураций процессора определим фактический размер кэш-памяти

Кэш 1-ого уровня (L1): 262144 bytes

Кэш 2-ого уровня (L2): 1048576 bytes

Кэш 3-ого уровня (L3): 8388608 bytes

Из графика видно, что первый скачок “случайной” кривой происходит как раз при размере массива около 262144 байтов: весь массив перестает помещаться в кэш первого уровня и поэтому часть элементов будет в кэше второго уровня, из-за чего будут происходит кэш-промахи и увеличиваться среднее время доступа. Так как обход случайный, то кэш-контроллер не сможет сделать эффективную предвыборку в кэш первого уровня и придется обращаться в другие уровни. Затем все большая часть массива будет находится в кэше второго уровня => время еще растет.

Следующий скачок происходит на 2097152 байтах. Часть массива начинает находиться в кэше третьего уровня и время обращения увеличивается. Т.е. размер второго уровня кэша точно меньше 2097152 байт

# ЗАКЛЮЧЕНИЕ

В ходе работы были исследованы зависимость времени доступа к данным в памяти от их объема, а также зависимость времени доступа к данным в памяти от порядка их обхода. Был построен график зависимости и полученные данные были сравнены с реальными значениями размера кэш-памяти.