###### 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, 18205 группы

Гайдамака Андрея Владиславовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А. Ю. Власенко

Новосибирск 2019

ЦЕЛЬ 3

ЗАДАНИЕ 3

ОПИСАНИЕ РАБОТЫ 3

ЗАКЛЮЧЕНИЕ 4

Приложение 1. Код программы 5

# 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

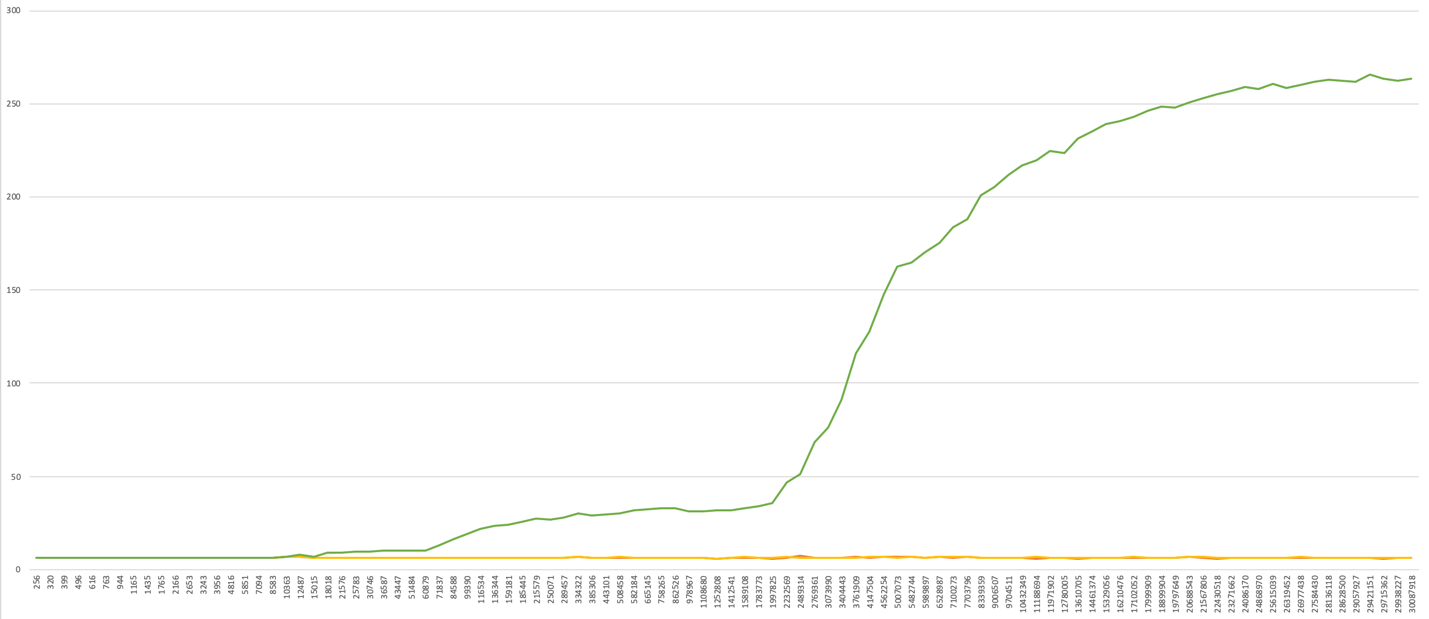
1. Написать программу, многократно выполняющую обход массива заданного размера тремя способами.
2. Для каждого размера массива и способа обхода измерить среднее время доступа к одному элементу (в тактах процессора). Построить графики зависимости среднего времени доступа от размера массива.
3. На основе анализа полученных графиков:

• определить размеры кэш-памяти различныхуровней,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

В ходе лабораторной работы было произведено заполнение массива тремя разными способами: прямым, обратным и случайным (по алгоритму Фишера-Йетса). Так же был произведен замер времени доступа к элементу в тактах для каждого вида массива при разных размерах массива. Замеры проводились для 100 разных размеров чтобы определить размеры кэш-памяти различных уровней. Все полученные значения записывались в csv файл и по этим данным был построен график зависимости количества тактов для доступа к элементу, для различных способов заполнения массива, от размера массива.



# ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были определены размеры кэш памяти которые совпадают с информацией о размерах кэш памяти на данном компьютере. Результаты показали, что время доступа к элементам массива с прямым и обратным обходом примерно равны, так как легко осуществляется работа аппаратной предвыборки данных, а в массиве со случайным обходом время доступа к элементу массива растет пропорционально размеру массива

**Приложение 1. Код программы**

#include <iostream>  
#include <limits>  
#include <fstream>  
  
  
using ull = unsigned long long;  
  
inline ull rdtsc() {  
 unsigned int lo, hi;  
 asm volatile ( "rdtsc\n" : "=a" (lo), "=d" (hi) );  
 return ((ull)hi << 32) | lo;  
}  
  
void direct\_pass\_fill(int \* arr, const int N){  
 for (int i = 0; i < N; ++i){  
 arr[i] = i + 1;  
 }  
 arr[N-1] = 0;  
}  
  
void return\_pass\_fill(int \*arr, const int N){  
 for (int i = N-1; i > 0; --i){  
 arr[i] = i - 1;  
 }  
 arr[0] = N-1;  
}  
  
void random\_pass\_fill(int \* arr, const int N){  
 direct\_pass\_fill(arr, N);  
 srand(time(NULL));  
 for (int i = N-1; i >= 1; --i){ //Алгоритм Фишера – Йетса  
 int j = rand() % (i+1);  
  
 int tmp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = tmp;  
 }  
}  
  
double count\_time(const int \* arr, const int size){  
 for (int i = 0, k =0; i < size; ++i){  
 k = arr[k];  
 }  
 ull min\_count = std::numeric\_limits<ull>::max();  
 for (int j = 0; j < 10; ++j){  
 ull start = rdtsc();  
 for (int i = 0, k = 0; i < size; ++i){  
 k = arr[k];  
 }  
 ull end = rdtsc();  
 if (end - start < min\_count){  
 min\_count = end - start;  
 }  
 }  
 return static\_cast<double>(min\_count) / size;  
}  
  
template <class T>  
void print\_csv\_array(std::ofstream& csv\_file, const T \* arr, int size){  
 for (int i = 0; i < size; ++i){  
 csv\_file << arr[i] << ';';  
 }  
 csv\_file << std::endl;  
}  
  
  
  
int main() {  
 const int NUMBER\_OF\_POINTS = 100;  
 double k = 0;  
 size\_t size = 256;  
 size\_t sizes[100];  
 double direct[100];  
 double reverse[100];  
 double random[100];  
 for (int i = 0; i < NUMBER\_OF\_POINTS; ++i){  
 int \*arr = new int[size];  
 sizes[i] = size;  
 direct\_pass\_fill(arr, size);  
 direct[i] = count\_time(arr, size);  
 return\_pass\_fill(arr, size);  
 reverse[i] = count\_time(arr, size);  
 random\_pass\_fill(arr, size);  
 random[i] = count\_time(arr, size);  
 size = static\_cast<size\_t>(size \* (1.25 - k));  
 k += 0.0025;  
 delete [] arr;  
 std::cout << i << std::endl;  
 }  
 std::ofstream csv\_file("out.csv");  
 if (!csv\_file.is\_open()){  
 throw std::runtime\_error("Cant open csv file");  
 }  
 csv\_file.setf(std::ios\_base::fixed);  
 csv\_file.precision(2);  
 print\_csv\_array(csv\_file, sizes, NUMBER\_OF\_POINTS);  
 print\_csv\_array(csv\_file, direct, NUMBER\_OF\_POINTS);  
 print\_csv\_array(csv\_file, reverse, NUMBER\_OF\_POINTS);  
 print\_csv\_array(csv\_file, random, NUMBER\_OF\_POINTS);  
 return 0;  
}