**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**

«Исследование влияния кэш-памяти на среднее время доступа к элементу массива»

студента 2 курса, группы 20203

**Синюкова Валерия Константиновича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент кафедры параллельных вычислений
Власенко Андрей Юрьевич

Новосибирск 2021

# СОДЕРЖАНИЕ

ЦЕЛЬ ..... 3
ЗАДАНИЕ ..... 3
ОПИСАНИЕ РАБОТЫ ..... 4
ЗАКЛЮЧЕНИЕ ..... 6
Приложение 1. *Листинг программы, реализующий обход массива тремя различными способами* ..... 7
Приложение 2. *Результирующая таблица исследования зависимости времени обхода массива от размера массива и способа обхода* ..... 9

# 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 (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.
4. На основе анализа 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.
5. Составить отчет по лабораторной работе. Отчет должен содержать следующее.
   - Описание алгоритмов заполнения массива тремя способами (особенно - случайным).
   - График и таблицу зависимости среднего времени доступа к одному элементу от размера массива и способов обхода.
   - Полный компилируемый листинг реализованной программы и команду для ее компиляции.
   - Вывод по результатам лабораторной работы.

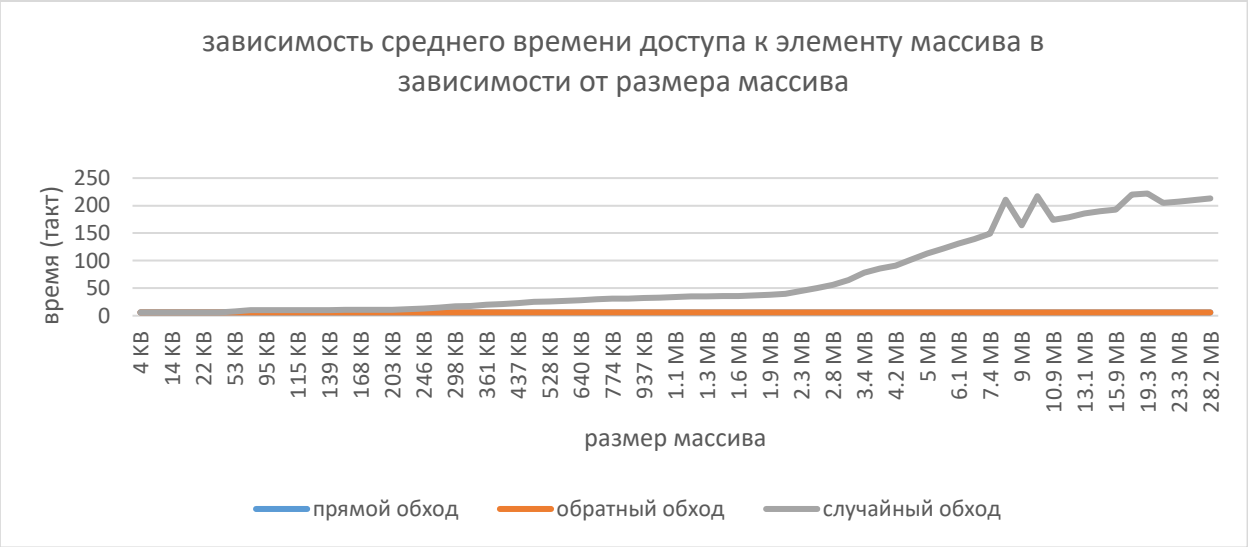
# ОПИСАНИЕ РАБОТЫ

1. Была написана программа на языке C++, многократно выполняющая обход массива целочисленных значений заданного размера тремя способами (прямой, обратный и случайный).
   Сам обход происходит следующим образом:
   ```
   k = 0;
   for (i = 0; i < N * K; ++i)
     k = x[k];
   ```
   N – количество элементов массива, K – количество обходов массива, x - массив.
   Заполнение массива в зависимости от обхода выглядит следующим образом:
   *Прямой обход:*
   `x[0] = 1, x[1] = 2, …, x[N-1] = 0.`
   *Обратный обход:*
   `x[0] = N-1, x[1] = 0, …, x[N-1] = N-2.`
   *Случайный обход:*
   элементы массива заполняются случайным образом.

   В начале программы выполняется некоторая подготовительная работа (а именно вычисление числа Пи с помощью разложения в ряд Грегори-Лейбница с количеством членов ряда, участвующих в разложении, равным $10^6$), это сделано для того, чтобы процессор с динамически изменяемой частотой установил ее на фиксированном уровне.
   Перед каждым обходом массива выполняется предварительный обход без замера времени для того, чтобы «прогреть кэш», то есть выгрузить из него все посторонние данные и по возможности поместить в него элементы массива.
   Программа был скомпилирована с помощью GNU C++ с уровнем оптимизации O1.
   После очередного обхода цикла происходит сравнение значения переменной k с константой, в случае равенства происходит вывод в поток: `cout << "!!!!";`, это сделано для того, чтобы в процессе оптимизации обход массива не был удален.
   Листинг данной программы можно посмотреть в соответствующем приложении.

2. Начальный размер массива был выбран заведомо меньше, чем размер кэша первого уровня, а именно 256 элементов, его объем, соответственно, равен 1 Кбайт (так как массив хранит данные типа int).
   С помощью машинной команды rdtsc измерялось среднее время доступа к элементу массива за обход (в тактах процессора). Для каждого варианта обхода было выполнено 20 измерений (для того, чтобы снизить влияние посторонних процессов), из которых для дальнейшего построения зависимости выбиралось наименьшее. Затем размер массива увеличивался в 1,2 раза и измерения выполнялись снова. Так размер массива

увеличивался в 1,2 раза по сравнению с предыдущим 109 раз. Таким образом, в последнем массиве, для которого выполнялось измерение, было 8157224 элементов (~31,1 Мбайт). На основе сделанных измерений был построен график (всю таблицу сделанных измерений можно посмотреть в соответствующем приложении).




Из данного графика и таблицы мы можем сделать вывод, что при прямом и обратном обходах при любом размере массива среднее время доступа к элементу массива будет постоянным, так происходит из-за блочной передачи данных в кэш первого уровня: обход элементов массива происходит в том же порядке, в каком они расположены в ОП, либо в обратном, из-за этого при доступе к первому элементу происходит копирование целого блока элементов, которые в ОП расположены рядом с прочитанным первым элементом.

Проанализируем результаты измерения времени при случайном обходе массива.
Пока размер массива был достаточно мал для того, чтобы все элементы могли разместиться в кэше первого уровня, среднее время доступа к элементу массива был точно такой же, как и в случае с прямым или обратным обходами.
Когда размер массива стал превышать размер кэша первого уровня, некоторые элементы массива были размещены в кэше второго уровня, из-за чего при доступе к ним происходил “кэш-промах”, следовательно, среднее время доступа к элементу массива увеличивалось пропорционально количеству элементов массива, которые не умещались в кэш первого уровня. Из графика можно сделать вывод, что размер кэша первого уровня примерно равен *30 Кбайт*.
Аналогичный «скачок» происходит, когда размер массива начинает превышать размер кэша второго уровня. Из графика можно сделать вывод, что размер кэша второго уровня примерно равен *250 Кбайт*.

Когда размер массива начинает превышать размер кэша третьего уровня, некоторые элементы становится необходимо загружать напрямую из ОП. Из-за этого, опять же, происходит увеличение среднего времени доступа к элементу массива пропорционально количеству элементов, которые не умещаются в кэш третьего уровня. Из графика можно сделать вывод, что размер кэша третьего уровня равен примерно *3 Мбайт*.

3. С помощью утилиты lscpu были точным образом определены размеры кэш-памяти:

```
L1d cache:                       32K
L1i cache:                       32K
L2 cache:                        256K
L3 cache:                        3072K
```

То есть размеры кэшей разных уровней, определенные с помощью анализа графика и таблицы из предыдущего пункта, были точно с учетом погрешности.

## ЗАКЛЮЧЕНИЕ

По результатам данной практической работы было измерено среднее время доступа к элементу массива в зависимости от размера массива, было изучено влияние кэш-памяти на это время при трех различных обходах массива: прямом, обратом и случайном. Были примерно измерены размеры кэш-памяти разных уровней, результаты этих измерений были сравнены с реальными размерами.

**Приложение 1.** *Листинг программы, реализующий обход массива тремя различными способами*

```
#define M 1000000
#define NUMBER_OF_MEASUREMENTS 20
#define NUMBER_OF_INCREASES 110
#define K 10

#include "include/lab1.h"
#include <fstream>
#include <iostream>
#include <ctime>
#include <cmath>
#include <cstdlib>
#include <stdint.h>

using namespace std;

void bypass(ofstream& out, int* x, int N)
{
	int j, i, k = 0;
	uint64_t t, min = 0;
	uint64_t tbegin, tend;
	uint32_t t1begin, t1end, t2begin, t2end;
	for (i = 0; i < N * K; ++i)
	{
		k = x[k];
	}
	if (123 == k)
		cout << "!!!!";
	for (j = 0; j < NUMBER_OF_MEASUREMENTS; ++j)
	{
		tbegin = __builtin_ia32_rdtsc();
		for (i = 0; i < N * K; ++i)
			k = x[k];
		tend = __builtin_ia32_rdtsc();
		if (123 == k)
			cout << "!!!!";
		t = (uint64_t)((tend - tbegin) / (N * K));
		if ((0 == min) || (t < min))
			min = t;
	}
	out << min << ',';
}

void straight(ofstream& out, int* x, int N)
{
	int i;
	for (i = 0; i < N - 1; ++i)
		x[i] = i + 1;
	x[N - 1] = 0;
	bypass(out, x, N);
}

void reverse(ofstream& out, int* x, int N)
{
```

```
	int i;
	for (i = N - 1; i > 0; --i)
		x[i] = i - 1;
	x[0] = N - 1;
	bypass(out, x, N);
}

void random(ofstream& out, int* x, int N)
{
	int i, k = 0, offset = 1, numberOfUnvisitedElements = N,
beginElementsChecked = 1, endElementsChecked = 0;
	bool* visited = new bool[N];
	for (i = 1; i < N; ++i)
		visited[i] = false;
	visited[0] = true;
	i = 0;
	while (numberOfUnvisitedElements != 1)
	{
		k = (rand() % (N - endElementsChecked - beginElementsChecked)) +
beginElementsChecked;
		while (visited[k] != false)
		{
			k += offset;
			if (k < 0)
				k += N;
			if (k > N - 1)
				k = k % N;
			offset = (int)pow(-1, offset) * (1 + abs(offset));
		}
		offset = 1;
		visited[k] = true;
		x[i] = k;
		i = k;
		if ((k == N - endElementsChecked - 1) && (numberOfUnvisitedElements
!= 2))
			while (visited[k] != false)
			{
				--k;
				++endElementsChecked;
			}
		if ((k == beginElementsChecked) && (numberOfUnvisitedElements != 2))
			while (visited[k] != false)
			{
				++k;
				++beginElementsChecked;
			}
		--numberOfUnvisitedElements;
	}
	x[k] = 0;
	delete[] visited;
	bypass(out, x, N);
}

int main()
{
	srand((unsigned)time(0));
	int N = 256, i, * x, space;
	double Pi = LeibnizFormula(M);
	time_t time;
```

```
    if (1.1 == Pi)
        cout << "!!!!";
    ofstream out("out.csv");
    for (i = 0; i < NUMBER_OF_INCREASES; ++i)
    {
        out << N << ',';
        space = round((N * 10) / 256);
        if (space < 10240)
            out << (double)(space / 10) << " KB,";
        else
            out << (double)(round(space / 1024) / 10) << " MB,";
        x = new int[N];
        straight(out, x, N);
        reverse(out, x, N);
        random(out, x, N);
        delete[] x;
        N = (int)(N * 1.2);
        out << endl;
    }
    return 0;
}
```

**Приложение 2.** *Результирующая таблица исследования зависимости времени обхода массива от размера массива и способа обхода*

| кол-во элементов массива | размер массива | прямой обход (такт) | обратный обход (такт) | случайный обход (такт) |
|---|---|---|---|---|
| 256 | 1 KB | 6 | 6 | 6 |
| 281 | 1 KB | 6 | 6 | 6 |
| 309 | 1 KB | 6 | 6 | 6 |
| 339 | 1 KB | 6 | 6 | 6 |
| 372 | 1 KB | 6 | 6 | 6 |
| 409 | 1 KB | 6 | 6 | 6 |
| 449 | 1 KB | 6 | 6 | 6 |
| 493 | 1 KB | 6 | 6 | 6 |
| 542 | 2 KB | 6 | 6 | 6 |
| 596 | 2 KB | 6 | 6 | 6 |
| 655 | 2 KB | 6 | 6 | 6 |
| 720 | 2 KB | 6 | 6 | 6 |
| 792 | 3 KB | 6 | 6 | 6 |
| 871 | 3 KB | 6 | 6 | 6 |
| 958 | 3 KB | 6 | 6 | 6 |
| 1053 | 4 KB | 6 | 6 | 6 |
| 1158 | 4 KB | 6 | 6 | 6 |
| 1273 | 4 KB | 6 | 6 | 6 |
| 1400 | 5 KB | 6 | 6 | 6 |
| 1540 | 6 KB | 6 | 6 | 6 |
| 1694 | 6 KB | 6 | 6 | 6 |
| 1863 | 7 KB | 6 | 6 | 6 |
| 2049 | 8 KB | 6 | 6 | 6 |

| 2253 | 8 KB | 6 | 6 | 6 |
|---|---|---|---|---|
| 2478 | 9 KB | 6 | 6 | 6 |
| 2725 | 10 KB | 6 | 6 | 6 |
| 2997 | 11 KB | 6 | 6 | 6 |
| 3296 | 12 KB | 6 | 6 | 6 |
| 3625 | 14 KB | 6 | 6 | 6 |
| 3987 | 15 KB | 6 | 6 | 6 |
| 4385 | 17 KB | 6 | 6 | 6 |
| 4823 | 18 KB | 6 | 6 | 6 |
| 5305 | 20 KB | 6 | 6 | 6 |
| 5835 | 22 KB | 6 | 6 | 6 |
| 6418 | 25 KB | 6 | 6 | 6 |
| 7059 | 27 KB | 6 | 6 | 6 |
| 7764 | 30 KB | 6 | 6 | 6 |
| 8540 | 33 KB | 6 | 6 | 6 |
| 9394 | 36 KB | 6 | 6 | 6 |
| 10333 | 40 KB | 6 | 6 | 7 |
| 11366 | 44 KB | 6 | 6 | 7 |
| 12502 | 48 KB | 6 | 6 | 8 |
| 13752 | 53 KB | 6 | 6 | 8 |
| 15127 | 59 KB | 6 | 6 | 8 |
| 16639 | 64 KB | 6 | 6 | 9 |
| 18302 | 71 KB | 6 | 6 | 9 |
| 20132 | 78 KB | 6 | 6 | 9 |
| 22145 | 86 KB | 6 | 6 | 10 |
| 24359 | 95 KB | 6 | 6 | 10 |
| 26794 | 104 KB | 6 | 6 | 10 |
| 29473 | 115 KB | 6 | 6 | 10 |
| 32420 | 126 KB | 6 | 6 | 10 |
| 35662 | 139 KB | 6 | 6 | 10 |
| 39228 | 153 KB | 6 | 6 | 11 |
| 43150 | 168 KB | 6 | 6 | 11 |
| 47465 | 185 KB | 6 | 6 | 11 |
| 52211 | 203 KB | 6 | 6 | 11 |
| 57432 | 224 KB | 6 | 6 | 12 |
| 63175 | 246 KB | 6 | 6 | 13 |
| 69492 | 271 KB | 6 | 6 | 15 |
| 76441 | 298 KB | 6 | 6 | 17 |
| 84085 | 328 KB | 6 | 6 | 18 |
| 92493 | 361 KB | 6 | 6 | 20 |
| 101742 | 397 KB | 6 | 6 | 21 |
| 111916 | 437 KB | 6 | 6 | 23 |
| 123107 | 480 KB | 6 | 6 | 25 |
| 135417 | 528 KB | 6 | 6 | 26 |
| 148958 | 581 KB | 6 | 6 | 27 |
| 163853 | 640 KB | 6 | 6 | 28 |

| | | | | |
|---|---|---|---|---|
| 180238 | 704 KB | 6 | 6 | 30 |
| 198261 | 774 KB | 6 | 6 | 31 |
| 218087 | 851 KB | 6 | 6 | 31 |
| 239895 | 937 KB | 6 | 6 | 32 |
| 263884 | 1 MB | 6 | 6 | 33 |
| 290272 | 1.1 MB | 6 | 6 | 34 |
| 319299 | 1.2 MB | 6 | 6 | 35 |
| 351228 | 1.3 MB | 6 | 6 | 35 |
| 386350 | 1.4 MB | 6 | 6 | 36 |
| 424985 | 1.6 MB | 6 | 6 | 36 |
| 467483 | 1.7 MB | 6 | 6 | 37 |
| 514231 | 1.9 MB | 6 | 6 | 38 |
| 565654 | 2.1 MB | 6 | 6 | 40 |
| 622219 | 2.3 MB | 6 | 6 | 45 |
| 684440 | 2.6 MB | 6 | 6 | 50 |
| 752884 | 2.8 MB | 6 | 6 | 56 |
| 828172 | 3.1 MB | 6 | 6 | 65 |
| 910989 | 3.4 MB | 6 | 6 | 78 |
| 1002087 | 3.8 MB | 6 | 6 | 86 |
| 1102295 | 4.2 MB | 6 | 6 | 91 |
| 1212524 | 4.6 MB | 6 | 6 | 102 |
| 1333776 | 5 MB | 6 | 6 | 113 |
| 1467153 | 5.5 MB | 6 | 6 | 122 |
| 1613868 | 6.1 MB | 6 | 6 | 131 |
| 1775254 | 6.7 MB | 6 | 6 | 139 |
| 1952779 | 7.4 MB | 6 | 6 | 149 |
| 2148056 | 8.1 MB | 6 | 6 | 211 |
| 2362861 | 9 MB | 6 | 6 | 164 |
| 2599147 | 9.9 MB | 6 | 6 | 217 |
| 2859061 | 10.9 MB | 6 | 6 | 174 |
| 3144967 | 11.9 MB | 6 | 6 | 179 |
| 3459463 | 13.1 MB | 6 | 6 | 186 |
| 3805409 | 14.5 MB | 6 | 6 | 190 |
| 4185949 | 15.9 MB | 6 | 6 | 193 |
| 4604543 | 17.5 MB | 6 | 6 | 220 |
| 5064997 | 19.3 MB | 6 | 6 | 222 |
| 5571496 | 21.2 MB | 6 | 6 | 205 |
| 6128645 | 23.3 MB | 6 | 6 | 207 |
| 6741509 | 25.7 MB | 6 | 6 | 210 |
| 7415659 | 28.2 MB | 6 | 6 | 213 |
| 8157224 | 31.1 MB | 6 | 6 | 217 |