###### 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

«Лабораторная работа 8»

Студента 1 курса, 19210 группы

**Пирожков Андрей Константинович**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

(ученая степень, звание)

Д.С.Иванишкин

Новосибирск 2020

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc58682233)

[ЗАДАНИЕ 3](#_Toc58682234)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc58682235)

[ЗАКЛЮЧЕНИЕ 7](#_Toc58682236)

[Приложение *Листинг файла Lab8.c* 8](#_Toc58682237)

# ЦЕЛЬ

* Исследование зависимости времени доступа к данным в памяти от их объема.
* 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. Каждый вариант программы оптимизировать по скорости, насколько это возможно.
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.
4. Составить отчет по лабораторной работе:
   * Титульный лист.
   * Цель лабораторной работы.
   * Описание способа заполнения массива тремя способами.
   * Графики 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**

Я написал одну программу, которая выводит все нужную мне статистику практически готовым графиком. Достаточно просто скопировать и вставить в csv файл.

В моей программе реализованы три функции, которые создают нужную структуру массива. Также имеется функция, которая совершает обход. И есть функция, которая позволяет замерять массивы не только размерами являющиеся степенью двойки, но и промежуточные. Сделал я это для того, чтобы замерить такты при 3МБ, ведь это размер кэш-памяти 3-его уровня.

В ходе реализации программы заметил, что счётчик команд, который был приведён в примере лабораторной работы №1 работает неадекватно, как только переменная типа int переполняется. Она заполняла первые 32 бита единичками, а следующие 32 бита начинали считаться заново. Из-за этого результаты внезапно после 8МБ вырастали в миллиарды раз. Разрешилась это проблема просто: я нашел в интернете другую функцию, которая тоже способна считывать такты процессора.

Также из-за оптимизации O1 мне пришлось создавать возвращаемые значения после каждого обхода и куда-то их копить, чтобы компилятор не счёл его за мертвый код. Я просто организовал двумерный массив, куда заносил результаты, а затем просто выводил его сумму перед завершением программы.

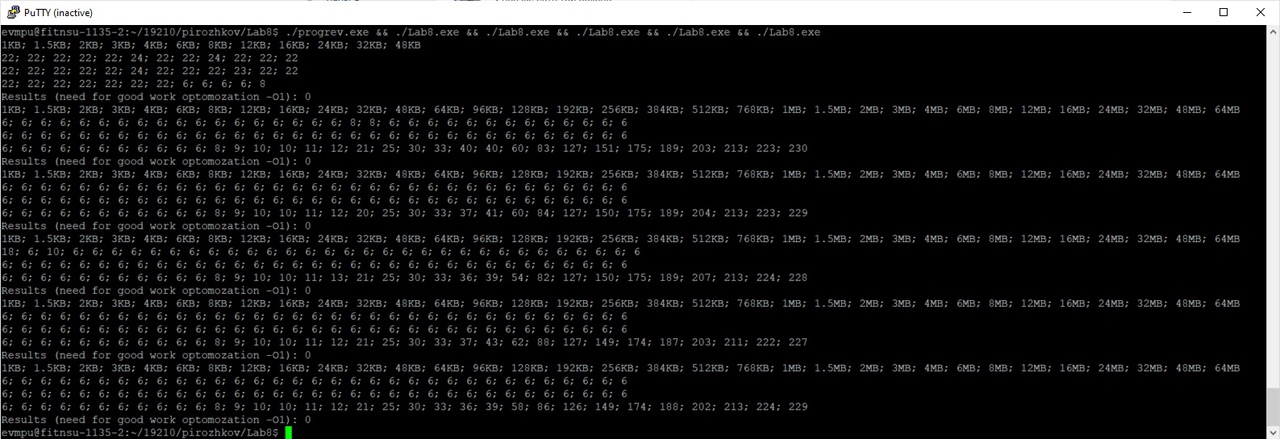
Также программа нуждалась в прогреве кэша. Причём его недостаточно просто одним циклом прогреть. Для этого лучше всего было сделать копию этой программы, которая бы прогонялось на более малые размеры массива. Так я и сделал.

Помимо этого, я не стал пренебрегать прогревом кэша перед каждым обходом массивов. На результат это вроде никак уже не влияло, однако в задании сказано делать именно так.

Компилировалась программа вот этой командой:

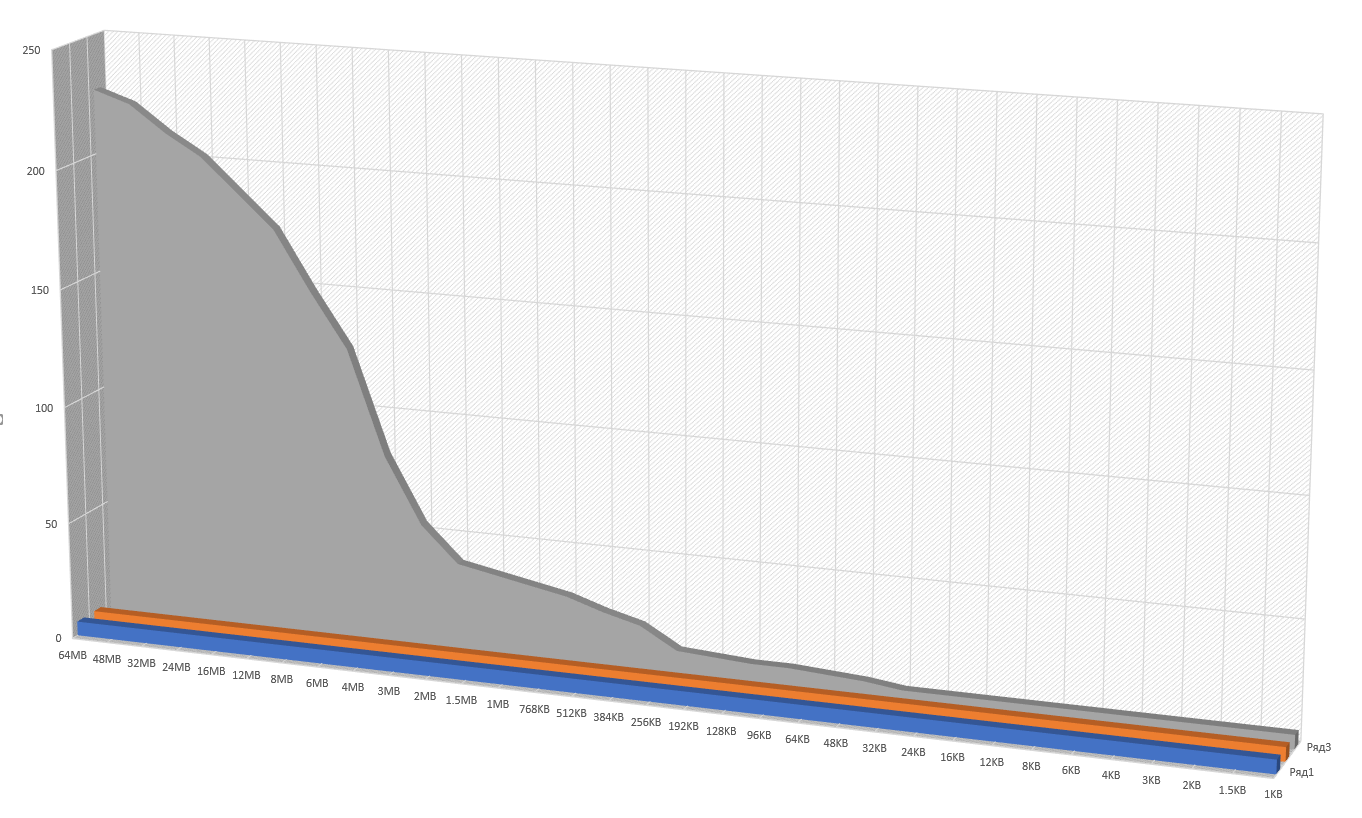
* gcc -Wall -O1 -o Lab8.exe Lab8.c

Прогонял свою программу несколько раз для выявления наименьших результатов:



Реальные значения размеров кэш-памяти легко узнать через консольную программку, которая оказалась внезапна установлена. Просто ввёл в терминал: «lstopo».

А теперь сам график, который у меня получился:



Смотреть надо справа налево. Сделал я так чтобы было виднее, на какой отметки начинается сильный рост графика. Сразу отмечу, что прямой и обратный обход в среднем получился равным 6 тактов на всём промежутке. Это связано с тем, что компилятор успевает подгадать с помощью предвыборки, какие значения будут следующие. И поэтому он без труда загружает целыми блоками в кэш-память следующие данные. Причем в кэш-память первого уровня. И скорее всего использует её в минимальных количествах, ведь можно бес конца перезаписывать. Её скорость составляет 6 тактов в секунду.

Теперь рассмотрим серый большой график. Это график рандомного обхода. Если внимательно присмотреться, то можно заметить, что первый скачок происходит после 32КБ. Это как раз кэш-память первого уровня. Просто она закончилась и часть от 48КБ перешла уже во второй уровень кэш-памяти. Предвыборка перестаёт работать и в итоге весь массив начинает заполнять всю кэш-память (в отличие от предыдущих, где благодаря предсказаниям было совсем не страшно потерять данные).

Далее можно видеть очень медленный рост. Он происходит, потому что всё больше информации лежит в кэш-памяти второго уровня. А как мы знаем, каждый следующий уровень кэш-памяти всё медленнее и медленнее. Далее идёт резкий скачок после 256КБ. Это означает, что тут закончилась память на втором уровне кэша.

Потом происходит всё тоже самое до 2МБ. Здесь можно заметить скачок. Он не большой, но если бы я не знал, что реальное значение это 3МБ, то я бы подумал, что кэш-память 3-его уровня ограничивается 2МБ. Однако, если посмотреть следующие скачки, то они всё равно больше, чем эти. Поэтому всё равно по графику можно предположить, что кэш-память 3-его уровня – 3МБ.

Прикреплю сюда таблицу для более точного изучения:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1КБ | 1.5КБ | 2КБ | 3КБ | 4КБ | 6КБ | 8КБ | 12КБ | 24КБ | 32КБ | 48КБ | 64КБ | 96КБ |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 8 | 9 | 10 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 128КБ | 192КБ | 256КБ | 384КБ | 512КБ | 768КБ | 1МБ | 1.5МБ | 2МБ | 3МБ | 4МБ | 6МБ |
| 10 | 11 | 12 | 21 | 25 | 30 | 33 | 36 | 39 | 54 | 82 | 126 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8МБ | 12МБ | 16МБ | 24МБ | 32МБ |
| 149 | 174 | 188 | 202 | 211 |

Можно сказать, что с реальными данными совпало (если не сильно обращать внимание на выбор между 2MB и 3MB):

* L1 – 32 KB
* L2 – 256 KB
* L3 – 3 MB

# ЗАКЛЮЧЕНИЕ

В ходе этой лабораторной работы я исследовал зависимость времени доступа к данным в памяти от их объёма, а также от их порядка. Я смог написать программу, которая исследовала эти зависимости всеми тремя способами, а также построил сравнительные диаграммы. Смогу определить размеры различных уровней кэш-памяти, и сопоставил их с реальными значениями, которые в принципе совпали.

# Приложение

*Листинг файла Lab8.c*

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 33

#ifdef \_\_i386

\_\_inline\_\_ unsigned long long rdtsc()

{

unsigned long long x;

\_\_asm\_\_ volatile ("rdtsc" : "=A" (x));

return x;

}

#elif \_\_amd64

\_\_inline\_\_ unsigned long long rdtsc()

{

unsigned long long a, d;

\_\_asm\_\_ volatile ("rdtsc" : "=a" (a), "=d" (d));

return (d << 32) | a;

}

#endif

void print(int\* M, int n)

{

for (int i = 0; i < n; i++)

{

printf("M[%d] = %d\n", i, M[i]);

}

printf("\n");

}

int\* do\_next(int\* M, int n)

{

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

{

M[i] = i + 1;

}

M[n - 1] = 0;

return M;

}

int\* do\_previous(int\* M, int n)

{

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

{

M[i] = i - 1;

}

M[0] = n - 1;

return M;

}

int\* do\_random(int\* M, int n)

{

for (int i = 0; i < n; i++)

{

M[i] = i;

}

srand((int)time(0));

int\* M\_ = (int\*)malloc(n \* sizeof(int));

int r = rand() % n;

int prev = r;

int start = r;

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

{

M[r] = -1;

r = rand() % n;

while ((M[r] == -1))

{

if (r != n - 1)

{

r++;

}

else

{

r = 0;

}

}

M\_[prev] = r;

prev = r;

}

M\_[prev] = start;

free(M);

return M\_;

}

int go(int\* M, int n, int k)

{

int a = 0;

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

{

a = M[a];

}

return a;

}

void get\_next\_n(int\* n, int\* q)

{

if (\*n % \*q == 0)

{

\*n = \*n + \*n / 2;

}

else

{

\*n = \*n / 3 \* 4;

\*q = \*n;

}

}

int main()

{

int\*\* results = (int\*\*)malloc(3 \* sizeof(int\*));

for (int i = 0; i < 3; i++)

{

results[i] = (int\*)malloc(N \* sizeof(int));

}

int r;

int k = 10;

int n = 256;

int q = n;

for (int i = 0; i < N; i++)

{

int c = n \* 4 / 1024;

if (c < 1024)

{

if (i != N - 1)

{

printf("%gKB; ", ((double)(n \* 4) / (double)(1024)));

}

else

{

printf("%gKB\n", ((double)(n \* 4) / (double)(1024)));

}

}

else

{

if (i != N - 1)

{

printf("%gMB; ", ((double)(n \* 4) / (double)(1024 \* 1024)));

}

else

{

printf("%gMB\n", ((double)(n \* 4) / (double)(1024 \* 1024)));

}

}

get\_next\_n(&n, &q);

}

n = 256;

q = n;

for (int i = 0; i < N; i++)

{

int\* M = (int\*)malloc(n \* sizeof(int));

M = do\_next(M, n);

r = go(M, n, 1);

unsigned long long time = rdtsc();

results[0][i] = go(M, n, k);

time = (rdtsc() - time) / (unsigned long long)(n \* k);

if (i != N - 1)

{

printf("%llu; ", time);

}

else

{

printf("%llu\n", time);

}

free(M);

get\_next\_n(&n, &q);

}

n = 256;

q = n;

for (int i = 0; i < N; i++)

{

int\* M = (int\*)malloc(n \* sizeof(int));

M = do\_previous(M, n);

r = go(M, n, 1);

unsigned long long time = rdtsc();

results[1][i] = go(M, n, k);

time = (rdtsc() - time) / (unsigned long long)(n \* k);

if (i != N - 1)

{

printf("%llu; ", time);

}

else

{

printf("%llu\n", time);

}

free(M);

get\_next\_n(&n, &q);

}

n = 256;

q = n;

for (int i = 0; i < N; i++)

{

int\* M = (int\*)malloc(n \* sizeof(int));

M = do\_random(M, n);

r = go(M, n, 1);

unsigned long long time = rdtsc();

results[2][i] = go(M, n, k);

time = (rdtsc() - time) / (unsigned long long)(n \* k);

if (i != N - 1)

{

printf("%llu; ", time);

}

else

{

printf("%llu\n", time);

}

free(M);

get\_next\_n(&n, &q);

}

r = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < N; j++)

{

r += results[i][j];

}

}

printf("Results (need for good work optomozation -O1): %d\n", r);

for (int i = 0; i < 3; i++)

{

free(results[i]);

}

free(results);

return 0;

}