###### 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 9»

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](#_Toc56767257)

[ЗАДАНИЕ 3](#_Toc56767258)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc56767259)

[ЗАКЛЮЧЕНИЕ 6](#_Toc56767260)

[Приложение *Листинг файла Lab9.c* 7](#_Toc56767261)

# ЦЕЛЬ

Экспериментальное определение степени ассоциативности кэш-памяти.

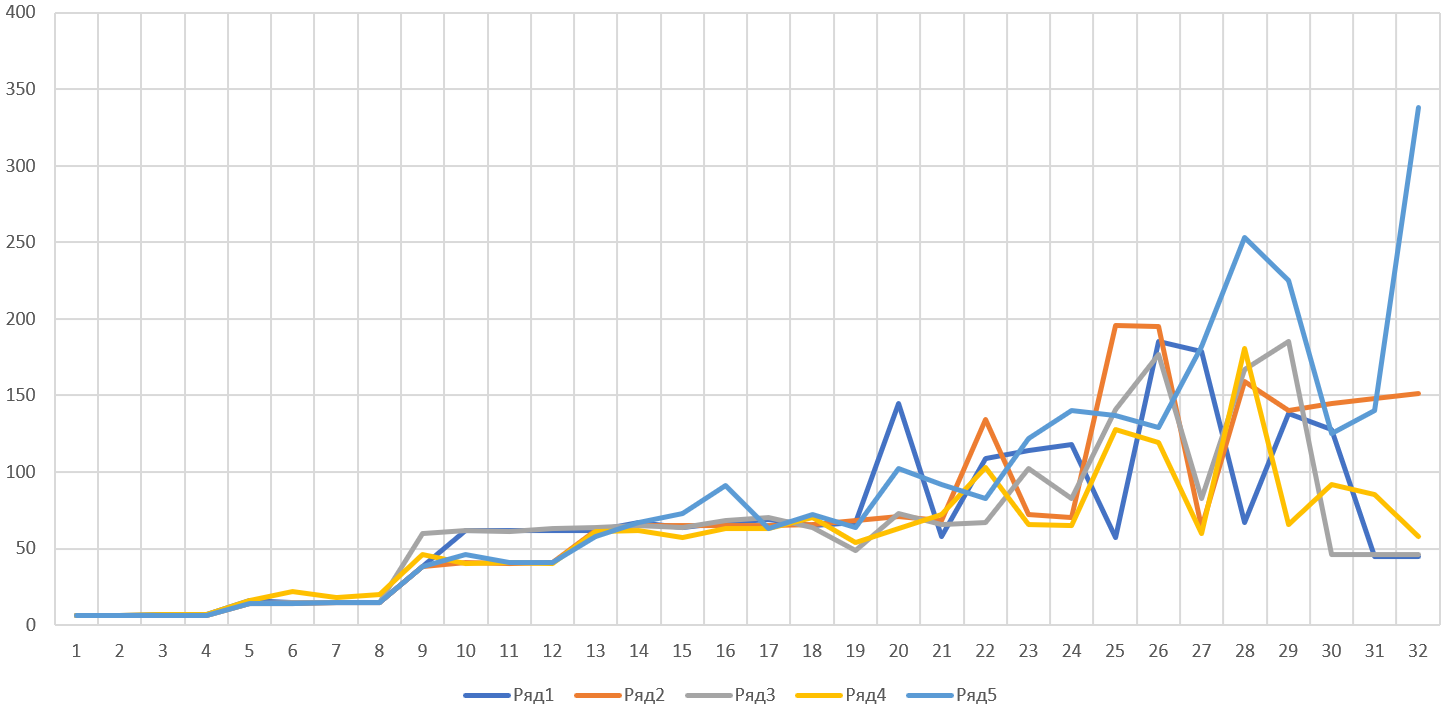
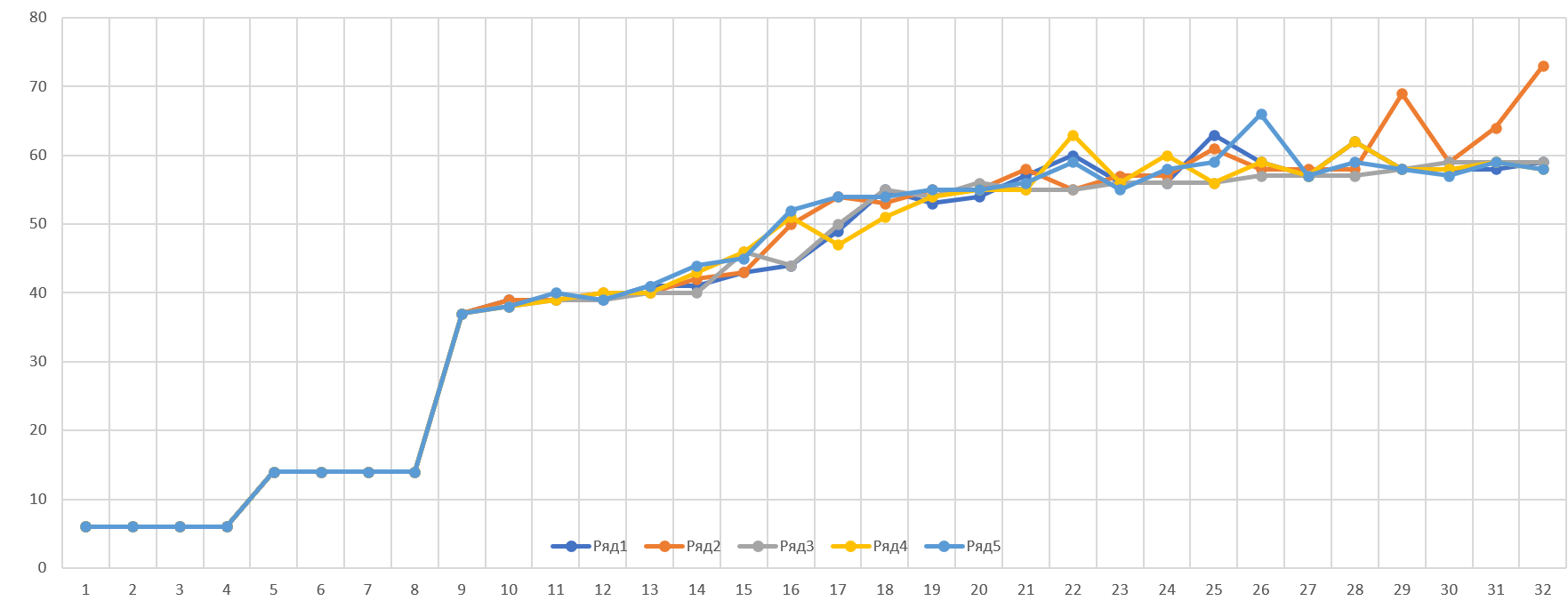
# ЗАДАНИЕ

1. Написать программу, выполняющую обход памяти в соответствии с заданием.
2. Измерить среднее время доступа к одному элементу массива (в тактах процессора) для разного числа фрагментов: от 1 до 32. Построить график зависимости времени от числа фрагментов.
3. По полученному графику определить степень ассоциативности кэшпамяти, сравнить с реальными характеристиками исследуемого процессора
4. Составить отчет по лабораторной работе. Отчет должен содержать следующие пункты:
   * Титульный лист.
   * Цель лабораторной работы.
   * Параметры теста: размер фрагментов, величина смещения.
   * График зависимости среднего времени доступа к элементу массива от числа фрагментов.
   * Реальные и полученные в ходе тестирования значения степени ассоциативности кэш-памятей процессора.
   * Полный компилируемый листинг реализованной программы и команды для ее компиляции.
   * Вывод по результатам лабораторной работы.

# **ОПИСАНИЕ РАБОТЫ**

В этой лабораторной работе я использовал некоторые исходные данные, а именно кэш-память, которая используется на сервере. Выяснил я это с помощью консольной команды «lstopo». Далее я отталкивался от этих 3 МБ.

Чтобы вызвать «буксование» памяти нужно было создать достаточно большой массив и правильное смещение. По рекомендации из задания выбрал интервал смещения между фрагментами равным 16МБ. Каждая итерация цикла соответствовала количеству фрагментов. Каждый фрагмент рассчитывался как размер кэша, делённый на количество фрагментов. Также на каждой итерации создавался массив специальным образом, как сказано в задании. А затем упорядоченный массив прогонялся и считались количество тактов.

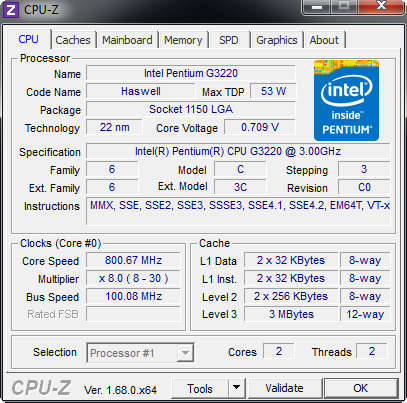
В ходе выполнения тестов выяснилось, что по моему графику не удаётся выяснить ассоциативность кэш-памяти 3-его уровня. Это оказалось довольно частой проблемой на сервере, причины которой я так и не понял. Я делал довольно много замеров и понял, что скачок после 12 скорее всего просто совпадение. Вот граф с тестами (1-ый скриншот):

Однако на следующий день результаты более интересны. Получился очень сильный разброс и нестабильность. В них примерно у половины видны скачки на 12. Также полно и других странных скачков. Вот так вот получилось (2-ой скриншот).

Использовал эту команду для компиляции (код программы в [Приложении](#_Приложение_1)):

* gcc -Wall -O1 -o Lab9.exe Lab9.c

Зато легко заметить кэш память буфера трансляции адресов – он равен 4. Кэш-память 1-ого и 2-ого уровня получилось 8. Всё это совпадает с реальными значениями. И степень ассоциативности 3-его уровня составляет 12, однако по моим графикам это сказать сложно, но зная процессор на сервере можно понять какая реальная ассоциативность:



# **ЗАКЛЮЧЕНИЕ**

В ходе этой лабораторной работы я написал программу, которая выполняет специальный обход памяти для подсчёта степени ассоциативности. Я измерил среднее время в тактах для разного числа фрагментов и получил соответствующие результаты, изображённые на диаграммах. Эксперимент получился частично успешным, однако всё-таки можно определить степень ассоциативности памяти.

# Приложение

*Листинг файла Lab9.c*

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define N 32

#define offset 4194304 //(16\*1024\*1024/4)

#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 do\_zero(int\* chache, int size)

{

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

{

chache[i] = 0;

}

}

void print\_fragments()

{

for (int i = 1; i <= 32; i++)

{

printf("%2d; ", i);

}

printf("\n");

}

void create\_massive(int\* chache, int block\_size, int blocks)

{

for (int i = 0; i < block\_size; i++)

{

int k = i;

for (int j = 0; j < blocks - 1; j++)

{

chache[k] = k + offset;

k += offset;

}

if (k + 1 != block\_size + offset \* (blocks - 1))

{

chache[k] = i + 1;

}

else

{

chache[k] = 0;

}

}

}

int go(int\* M, int n)

{

int k = 0;

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

{

k = M[k];

}

return k;

}

unsigned long long do\_go(int\* M, int n, int\* check)

{

unsigned long long time = rdtsc();

for (int i = 0; i < 50; i++)

{

\*check += go(M, n);

}

time = (rdtsc() - time) / (unsigned long long)(n\*50);

return time;

}

int main()

{

int result = 0;

int size\_L3 = 3072 \* 1024 / 4;

print\_fragments();

for (int k = 0; k < 10; k++)

{

for (int i = 1; i <= 32; i++)

{

int chache\_size = offset \* i;

int\* chache = (int\*)malloc(chache\_size \* sizeof(int));

do\_zero(chache, chache\_size);

create\_massive(chache, (size\_L3 / i), i);

printf("%2llu; ", do\_go(chache, (size\_L3 - (size\_L3 % i)), &result));

free(chache);

}

printf("%d \n", result);

}

return 0;

}