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»

студентки 2 курса, 24202 группы

Корсун Дарья Андреевна

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
В.А. Перепелки

Новосибирск 2025

СОДЕРЖАНИЕ

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 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.

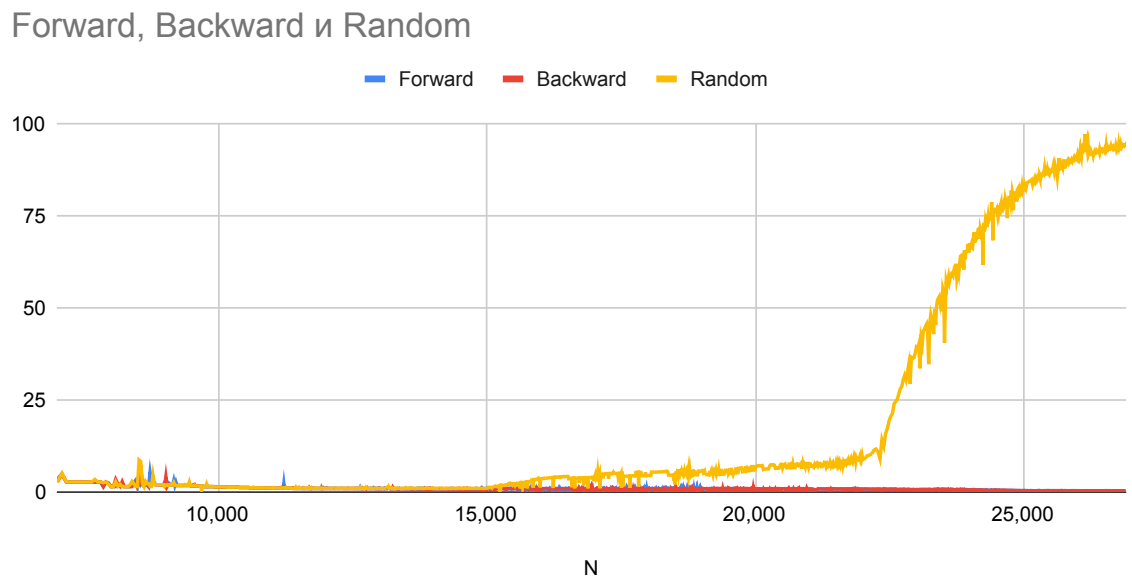
4. Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

В ходе работы были реализован код для измерения времени обхода массива для отслеживания влияния кэш-памяти на время работы программы. При этом в работе оценивается время доступа для разных уровней обхода, так как стоит задача узнать насколько кэш-контроллер способен распознать последовательный обход памяти и обеспечивают эффективную предварительную загрузку данных в кэш- память. Были написаны соотвествующие заполнения массивов: прямой, обратный и случайный.

Перед каждый подсчетом идет первичный проход по массивы с целью выгрузить из кэш-памяти посторонние данные, разместив там (по возможности) необходимые нам данные.

Шаг N изменяется по возрастанию, ведь разница в показаниях между ближайшими значениями становится не такая кардинальная. Помимо этого внесено динамическое изменение K так как для маленьких размеров массива важно много раз по нему итерироваться дабы получить более точные значения для каждого. Ось Ox представлена логарифмически по основанию 2 от размеров масива.



После анализа графика, получилось:
1) Размер L1 кэша данных – 128 КБ
2) Размер L2 кэша – 16 МБ
Что примерно совпадает с ожидаемыми значениями.

Код программы:

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

uint64_t read_tsc() {
  uint64_t val;
  asm volatile("mrs %0, cntvct_el0" : "=r"(val));
  return val;
}

void fill_forward(int* array, int N) {
  for (int i = 0; i < N; i++) array[i] = (i + 1) % N;
}

void fill_backward(int* array, int N) {
  for (int i = 0; i < N; i++) array[i] = (i == 0) ? N - 1 : i - 1;
}

void fill_random(int* array, int N) {
  for (int i = 0; i < N; i++) array[i] = i;
  for (int i = N - 1; i > 0; i--) {
    int j = rand() % (i + 1);
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
  }
}

void first_pass(int* array, int N) {
  int k = 0;
  for (int i = 0; i < N; i++) {
    k = array[k];
  }
  volatile int prevent_opt = k;
}

double full_pass(int* array, int N, int K) {
  int k = 0;
  uint64_t start = read_tsc();

  for (int i = 0; i < N * K; i++) {
    k = array[k];
  }

  uint64_t end = read_tsc();
  volatile int prevent_opt = k;

  return (double)(end - start) / (N * K);
}

int get_optimal_K(int N) {
  if (N <= 1024)
    return 100;
  else if (N <= 8192)
    return 50;
  else if (N <= 65536)
    return 20;
  else
    return 10;
}
```

```c
void generate_csv() {
  srand(time(0));
  FILE* file =
      fopen("/Users/dariakorsun/labs/Computer_labs/lab_8/results.csv", "w");
  printf("N,Forward,Backward,Random\n");
  fprintf(file, "N,Forward,Backward,Random\n");
  int flag = 0;
  for (int N = 128; N <= 2 * 32 * 256 * 1024;) {
    int* array = (int*)malloc(N * sizeof(int));
    if (array == NULL) {
      printf("Memory allocation failed for N=%d\n", N);
      break;
    }

    int K = get_optimal_K(N);

    fill_forward(array, N);
    first_pass(array, N);
    double forward_time = full_pass(array, N, K);

    fill_backward(array, N);
    first_pass(array, N);
    double backward_time = full_pass(array, N, K);

    fill_random(array, N);
    first_pass(array, N);
    double random_time = full_pass(array, N, K);

    fprintf(file, "%d,%.2f,%.2f,%.2f\n", N, forward_time, backward_time,
            random_time);
    // printf("%d,%.2f,%.2f,%.2f\n", N, forward_time, backward_time,

    free(array);

    if (N < 256) {
      if (flag == 0) {
        printf("256\n");
        flag = 1;
      }
      N += 4;
    } else if (N >= 256 && N < 512) {
      flag = 0;
      N += 4;
    } else if (N >= 512 && N < 1024) {
      if (flag == 0) {
        printf("1024\n");
        flag = 1;
      }
      N += 4;
    } else if (N >= 1024 && N < 2048) {
      flag = 0;
      N += 4;
    } else if (N >= 2048 && N < 4096) {
      if (flag == 0) {
        printf("4096\n");
        flag = 1;
      }
      N += 8;
    } else if (N >= 4096 && N < 8192) {
      flag = 0;
      N += 16;
    } else if (N >= 8192 && N < 16384) {
      if (flag == 0) {
        printf("16384\n");
        flag = 1;
      }
```

```
      N += 32;
    } else if (N >= 16384 && N < 32768) {
      flag = 0;
      N += 64;
    } else if (N >= 32768 && N < 65536) {
      if (flag == 0) {
        printf("65536\n");
        flag = 1;
      }
      N += 128;
    } else if (N >= 65536 && N < 131072) {
      flag = 0;
      N += 256;
    } else if (N >= 131072 && N < 262144) {
      if (flag == 0) {
        printf("262144\n");
        flag = 1;
      }
      N += 512;
    } else if (N >= 262144 && N < 524288) {
      flag = 0;
      N += 1024;
    } else if (N >= 524288 && N < 1048576) {
      if (flag == 0) {
        printf("1048576\n");
        flag = 1;
      }
      N += 2048;
    } else if (N >= 1048576 && N < 2097152) {
      flag = 0;
      N += 8192;
    } else if (N >= 2097152 && N < 4194304) {
      if (flag == 0) {
        printf("4194304\n");
        flag = 1;
      }
      N += 16384;
    } else if (N >= 4194304) {
      flag = 0;
      N += 32768;
    }
  }
}

int main() {
  generate_csv();
  return 0;
}
```

ЗАКЛЮЧЕНИЕ

Исследованы зависимости времени доступа к данным в памяти от их объема, исследованы зависимости времени доступа к данным в памяти от порядка их обхода.

На основе полученных данных были сделаны выводы о размере кэша разных уровней:

1) Размер L1 кэша данных – 128 КБ
2) Размер L2 кэша – 16 МБ