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

«ВЛИЯНИЕ КЭШ-ПАМЯТИ НА ВРЕМЯ ОБРАБОТКИ МАССИВОВ»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2022

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 (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 (на основе документации по процессору используемой машины; утилите, отражающей характеристики процессора; системному файлу;…)

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

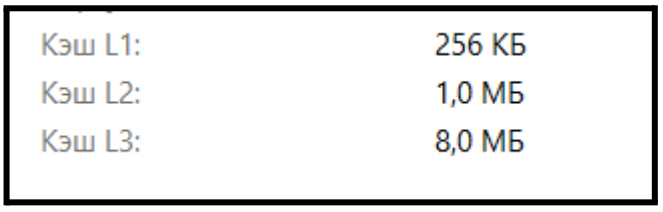
1) Перед запуском самой программы около 1 секунды был выполнен прогон пустого цикл с расчётом умножение матриц(из прошлой лабораторной работы), чтобы процессор с динамически изменяемой частотой установил её на фиксированном уровне.
2) Для каждого размера массива и способа обхода было измерено среднее время доступа к одному элементу (в тактах процессора)
3) Описание обходов:
    a) Прямой

       Значением ячейки массива с индексом i будет i+1. В случае последнего элемента следующим элементом будет самый первый
    b) Обратный

       То же самое, что и прямой обход, но теперь обход идёт с конца массива
    c) Случайный

       Значением ячейки массива с индексом i будет некоторый индекс, который невозможно предугадать заранее, но индекс не выйдет за пределы массива.

4) Были построены графики зависимости среднего времени доступа от размера массива
5) Были определены размеры кэш-памяти точным образом. Сравнив с графиков, мы получили примерно одинаковые значения.



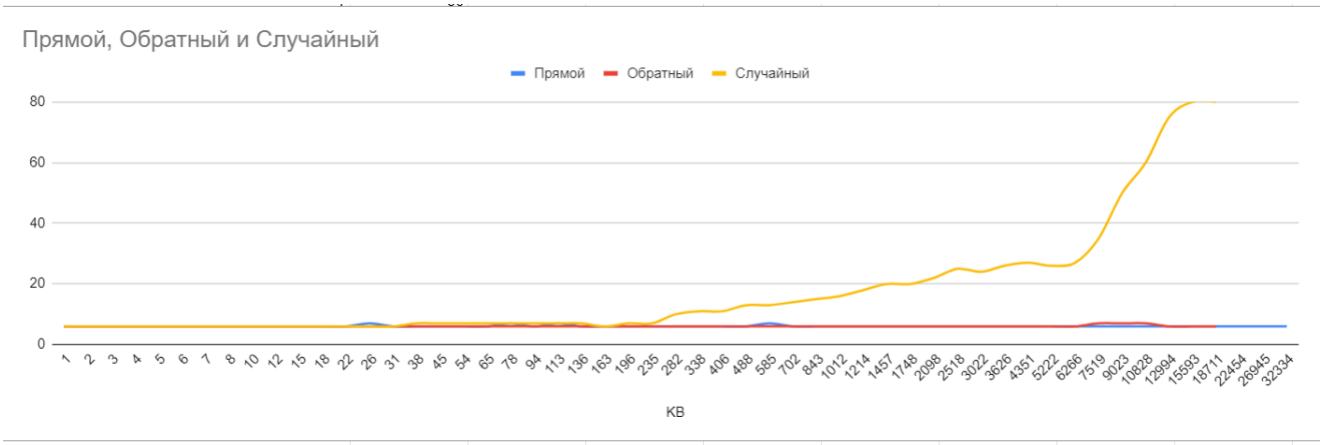
Заключение

Из-за предвыборки данных, прямой и обратный обходы не росли с увеличением размера массива.

В случайном же порядке количество тактов процессора, затраченное на получение элемента массива, заметно растет с увеличением размера массива потому что что кэш-контроллер не может корректно выполнить предвыборку данных для обхода массива в случайном порядке.

Приложение

Приложение 1. График зависимости среднего времени доступа от размера массива



Приложение 2. Полный листинг программы

```
1.  #include <iostream>
2.  #include <cstdlib>
3.  #include <ctime>
4.  #include <climits>
5.  #include <set>
6.  #include <string>
7.  #include <intrin.h>
8.  #pragma intrinsic(__rdtsc)
9.  #include <iomanip>
10. #include <vector>
11.
12. enum MODE
13. {
14.     DIRECT,
15.     REVERSE,
16.     RANDOM
17. };
18.
19. void directTraversal(unsigned int *array, const unsigned int n)
20. {
21.     array[n - 1] = 0;
22.     for (unsigned int i = 0; i < n - 1; i++)
23.     {
24.         array[i] = i + 1;
25.     }
26. }
27.
28. void reverseTraversal(unsigned int *array, const unsigned int n)
29. {
30.     array[0] = n - 1;
31.     for (unsigned int i = n - 1; i > 0; --i)
32.         array[i] = i - 1;
33. }
34.
35. void randomTraversal(unsigned int *array, const unsigned int arraySize)
36. {
37.     std::set<unsigned int> set;
38.
39.     for(unsigned int i = 0; i < arraySize; ++i) set.insert(i);
40.     unsigned int index = rand() % set.size();
41.     unsigned int prev = *next(set.begin(), index);
42.     unsigned int start = prev;
43.
```

```
    for(; set.size() > 1;)
    {
        set.erase(next(set.begin(), index));
        index = rand() % set.size();
        array[prev] = *next(set.begin(), index);
        prev = *next(set.begin(), index);

    }
    array[*set.begin()] = start;

}

void assign(const unsigned int *array, const unsigned int n, const unsigned int
    k)
{
    int m = 0;
    for (unsigned int j = 0; j < n * k; ++j)
        m = array[m];
}

void CalcTime(MODE modeName, const unsigned int minSize, const unsigned
    int maxSize,
              unsigned int *array, const int k)
{
    std::cout << std::setw(7)<< std::left << "KB"
              << std::setw(15)<< std::left << "size"
              << std::setw(7)<< std::left << "time" << std::endl;
    unsigned int currentSize = minSize;
    for (; currentSize < maxSize; currentSize = (unsigned int)(currentSize * 1.2))
    {
        array = new unsigned int[currentSize];

        if (modeName == DIRECT)           directTraversal(array, currentSize);
        else if (modeName == REVERSE)     reverseTraversal(array, currentSize);
        else                              randomTraversal(array, currentSize);

        unsigned long long minTime = LLONG_MAX;

        assign(array, currentSize, 3);

        for (int j = 0; j < 3; j++)
        {
            unsigned long long start = __rdtsc();
            assign(array, currentSize, k);
            unsigned long long end = __rdtsc();
            unsigned long long Time = end - start;
            if (Time < minTime) minTime = Time;
        }
```

```
90.
91.         std::cout << std::setw(7) << std::left << currentSize / 256
92.                   << std::setw(15)<< std::left << currentSize
93.                   << std::setw(7) << std::left << minTime / (k * currentSize) << std::endl;
94.
95.         free(array);
96.     }
97. }
98.
99. int main()
100.     {
101.         unsigned int minSize = 256;
102.         unsigned int maxSize = 32 * 1024 * 256;
103.         unsigned int k = 10;
104.         unsigned int *array;
105.
106.         std::cout << "DIRECT TRAVERSAL" << std::endl;
107.         CalcTime(DIRECT, minSize, maxSize, array, k);
108.
109.         std::cout << "REVERSE TRAVERSAL" << std::endl;
110.         CalcTime(REVERSE, minSize, maxSize, array, k);
111.
112.         std::cout << "RANDOM TRAVERSAL" << std::endl;
113.         CalcTime(RANDOM, minSize, maxSize, array, k);
114.
115.     }
```