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 21205

Евдокимовой Дари Евгеньевны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук, доцент
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: прямым, обратным, случайным.

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. На основе анализа полученных графиков:

   3.1. определить размеры кэш-памяти различных уровней, обосновать ответ, сопоставить результат с известными реальными значениями;

   3.2 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.

ОПИСАНИЕ РАБОТЫ

В ходе задания использовался компьютер с архитектурой amd64, с операционной системой Ubuntu 20.04.5 LTS и процессором Intel® Core™ i3-6100U CPU @ 2.30GHz × 4.

1. Была создана программа cache.cpp. Полный листинг представлен в Приложении 1.

   Команда для компиляции: *g++ -O1 cache.cpp -o cache*

   Команда для запуска: *./cache*
2. Бал написан метод прямого обхода массива, это линейный обход, при котором в ячейке будет лежать индекс следующей ячейки массива. У последнего элемента индекс будет нулевым, т.е. минимальным.
3. Бал написан метод обратного обхода массива, тот же самый способ обхода, что прямой, но в данном случае мы обходим массив с конца. У нулевого элемента индекс будет равен (количество элементов – 1), т.е. максимальным.
4. Бал написан метод случайного обхода массива. Он заключается в том, что на начальной итерации мы выбираем любое число, пусть это будет число0 (не выходящее за пределы массива) и в нулевую ячейку кладём это значение. Затем в ячейку с индексом числа, выбранного на предыдущей итерации (число0), мы кладем любое число, не равное числу0 и не равное индексу, который использовался (т.е. нулю). Таким образом мы гарантируем, что мы обойдем каждый элемент массива и не выйдем за его пределы.

   Пример: дан массив array[5]. В array[0] кладем любое из чисел [1-4]. Пусть это будет число 3. Получим array[0] = 3.

   Затем в array[3] мы кладём любое число из набора: 1, 2, 4. Пусть это будет число 1. Получим array[0] = 3, array[3] = 1.

   Затем в array[1] кладем число из набора: 2, 4. Пусть это будет число 2.

   Получим array[0] = 3, array[3] = 1, array[1] = 2.

   Затем в array[0] кладём число из набора: 4.

   Получим array[0] = 3, array[3] = 1, array[1] = 2, array[2] = 4.

   Затем в array[4] кладём 0 и получаем: array[0] = 3, array[3] = 1, array[1] = 2, array[2] = 4, array[4]=0.
5. Написанные методы обхода массива были проверены в методе CheckCorrectCalculating(), результаты представлены в Приложении 2.
6. В начале программы был запущен алгоритм перемножения матриц для того, чтобы процессор с линамически изменяемой частотой установил эту частоту на фиксированном уровне, в противном случае процессор будет работать на пониженной

частоте, возможны очень большие значения (порядка 6) в начале замера функци обращения к элементу массива, что нужно избежать, т.к. результат получим неверный.

7. По тем же причинам был проведен «прогрев кэша».
8. Компиляция проводилась на уровне –O1 для того, чтобы избежать обращений в оперативную память (на стек). Так же были добавлены дополнительные «выводы» в методах для того, чтобы компилятор не посчитал их «ненужными» для результата программы.
9. Количество проходов для большей точности равно 100. Количество тактов было измерено при помощи счётчика rdtsc() из библиотеки <x86intrin.h>.
10. Результаты программы представлены в Приложении 2. Для удобства построения графика была добавлена колонка KiB делением соответствующего значения на 256 из колонки Tics.
11. График по вышеприведенным результатам см. Рис. 1.

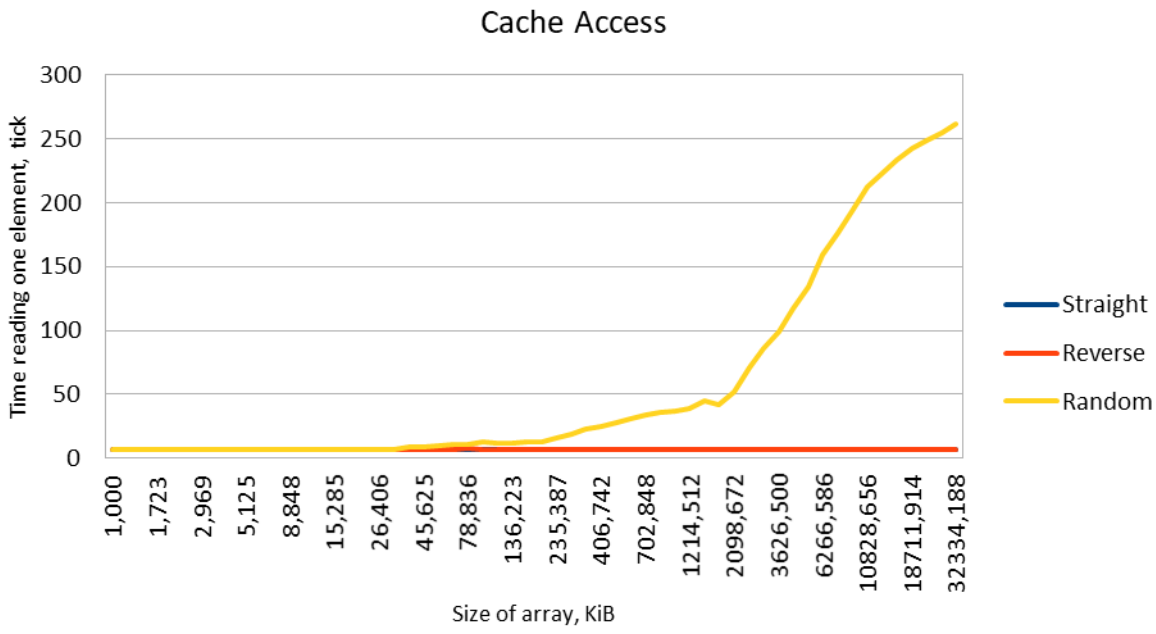

Рис 1. График зависисмоти времени доступа к элементу от размера и способов обхода

12. По графику можно заметить, что прирост тактов начинается с 32 kB, а это говорит о том, что размер кэша L1 равен 32 kB.
    Следующее возрастание начинается между ~260kB, что соответствует размеру кэша L2 – 216kB.
    Самый сильный скачок наблюдается после ~3000kB, что соответствует раземру кэша L3.
13. Реальные значения кэша получены с помощью команды –lscpu в Linux и программы CPU-Z в Windows. Данные, полученные с помощью этих программ подтверждают оценки кэша.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были выявлены зависимости времени доступа к данным в памяти от их объема и порядка их обхода. На основе полученных результатов можно сделать вывод о том, что время доступа почти одинаковое при прямом и обратном обходах и оно не зависит от размера массива, поскольку происходит предвыборка данных. А при случайном обходе время доступа к элементу массива увеличивается прямо пропорционально размеру массива.

Приложение 1. Листинг программы обращения к элементу массива

```
#include <ctime>
#include <iostream>
#include <x86intrin.h> //for __rdtsc()

/* ==================== CPU PREHEAT ==================== */
void PrintMatrix(int *matrix, size_t sizeMatr) {
  for (int i = 0; i < sizeMatr; i++) {
    for (int j = 0; j < sizeMatr; j++) {
      std::cout << matrix[i * sizeMatr + j] << " ";
      if (i % 5 == 0) {
        std::cout << "WoW I can print a matrix!" << std::endl;
      }
    }
    std::cout << std::endl;
  }
  std::cout << "=================================================="
            << std::endl;
}

void FillMatrix(int *matrix, size_t sizeMatr) {
  srand(time(NULL));
  for (size_t i = 0; i < sizeMatr * sizeMatr; i++) {
    matrix[i] = rand() % 50;
  }
}

void MultMatrixes(int *matrix1, int *matrix2, int *result,
                  size_t sizeMatr) {
  for (int i = 0; i < sizeMatr; i++) {
    for (int k = 0; k < sizeMatr; k++) {
      for (int j = 0; j < sizeMatr; j++) {
        result[i * sizeMatr + j] +=
            matrix1[i * sizeMatr + k] * matrix2[k * sizeMatr + j];
      }
    }
  }
}

void PrepareCPU() {
  size_t sizeMatr = 2500;

  int *m1 = new int[sizeMatr * sizeMatr]();
  FillMatrix(m1, sizeMatr);

  int *m2 = new int[sizeMatr * sizeMatr]();
  FillMatrix(m2, sizeMatr);

  int *resMatr = new int[sizeMatr * sizeMatr]();
```

```
  MultMatrixes(m1, m2, resMatr, sizeMatr);
  PrintMatrix(resMatr, sizeMatr / 2);
}

/* ==================== CACHE PREHEAT ==================== */
void PrepareCache(int *array, size_t lengthArr) {
  int elem = 0;
  for (size_t i = 0; i < lengthArr; i++) {
    elem = array[elem];
  }
  if (elem == 25) {
    std::cout << "Hello from 25 :) " << std::endl;
  }
}

/* ================= TYPES TO GO TROUGHT ARRAY ================= */
void GoStraight(int *array, size_t lengthArr) {
  for (size_t i = 0; i < lengthArr - 1; ++i) {
    array[i] = i + 1;
  }
  array[lengthArr - 1] = 0;
}

void GoReverse(int *array, size_t lengthArr) {
  for (size_t i = lengthArr - 1; i > 0; --i) {
    array[i] = i - 1;
  }
  array[0] = lengthArr - 1;
}

void GoRandom(int *array, size_t lengthArr) {
  srand(time(nullptr));
  bool *used = new bool[lengthArr];
  for (size_t i = 0; i < lengthArr; i++) {
    used[i] = false;
  }
  int j = 0;
  used[0] = true;
  for (size_t i = 0; i < lengthArr - 1; i++) {
    int tmp;
    while (used[tmp = rand() % lengthArr] != false)
      ;

    array[j] = tmp;
    used[tmp] = true;
    j = tmp;
  }
  array[j] = 0;
  delete[] used;
}
```

```cpp
/* ================= CHECK CORRECTNESS OF TYPES ================= */
void PrintArray(int *array, size_t lengthArr) {
  for (size_t i = 0; i < lengthArr; i++) {
    std::cout << array[i] << " ";
  }
  std::cout << std::endl;
}

void CheckCorrectCalculating() {
  size_t size = 20;
  std::cout << "size : " << size << std::endl;
  int *arr = new int[size]();

  GoStraight(arr, size);
  std::cout << "straight: ";
  PrintArray(arr, size);

  GoReverse(arr, size);
  std::cout << "reversed: ";
  PrintArray(arr, size);

  GoRandom(arr, size);
  std::cout << "random:   ";
  PrintArray(arr, size);
}

/* ================ COUNT TIME PER ELEMENT IN TICKS ================ */
double CountTimeInTicks(int *array, size_t lengthArr) {
  PrepareCache(array, lengthArr);

  size_t numberOfBypass = 100;

  double time;

  int elem = 0;
  for (int j = 0; j < numberOfBypass; j++) {
    double start = __rdtsc();
    for (int i = 0; i < lengthArr; i++) {
      elem = array[elem];
      if (elem == (lengthArr + 10))
        std::cout << "WooW, I'm out of bounds! =)" << std::endl;
    }
    double end = __rdtsc();

    time = end - start;
  }
  return (time / lengthArr);
}

int main() {
```

```
  CheckCorrectCalculating();
  PrepareCPU();

  std::cout << " ======= Straight bypass ======= " << std::endl;
  for (size_t sizeArr = 256; sizeArr < 256 * 32 * 1024;
       sizeArr *= 1.2) {
    std::cout << "Straight: ";
    std::cout << "Number: " << sizeArr << " ";
    int *array = new int[sizeArr];

    GoStraight(array, sizeArr);
    double time = CountTimeInTicks(array, sizeArr);
    std::cout << "Ticks: " << time << std::endl;
    delete[] array;
  }
  std::cout << std::endl << std::endl;

  std::cout << " ======= Reversed bypass ======= " << std::endl;
  for (size_t sizeArr = 256; sizeArr < 256 * 32 * 1024;
       sizeArr *= 1.2) {
    std::cout << "Reverse: ";
    std::cout << "Number: " << sizeArr << " ";
    int *array = new int[sizeArr];

    GoReverse(array, sizeArr);
    double time = CountTimeInTicks(array, sizeArr);
    std::cout << "Ticks: " << time << std::endl;
    delete[] array;
  }
  std::cout << std::endl << std::endl;

  std::cout << " ======= Random bypass ======= " << std::endl;
  for (size_t sizeArr = 256; sizeArr < 256 * 32 * 1024;
       sizeArr *= 1.2) {
    std::cout << "Random: ";
    std::cout << "Number: " << sizeArr << " ";
    int *array = new int[sizeArr];

    GoRandom(array, sizeArr);
    double time = CountTimeInTicks(array, sizeArr);

    std::cout << "Ticks: " << time << std::endl;
    delete[] array;
  }
  return 0;
}
```

Приложение 2. Проверка корректности.

```
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ g++ -O1 cache.cpp -o cache
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ ./cache
size : 5
straight: 1 2 3 4 0
reversed: 4 0 1 2 3
random:   3 4 0 1 2
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ g++ -O1 cache.cpp -o cache
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ ./cache
size : 9
straight: 1 2 3 4 5 6 7 8 0
reversed: 8 0 1 2 3 4 5 6 7
random:   5 2 8 7 1 3 4 6 0
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ g++ -O1 cache.cpp -o cache
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ ./cache
size : 15
straight: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 0
reversed: 14 0 1 2 3 4 5 6 7 8 9 10 11 12 13
random:   8 12 4 10 5 7 3 14 1 13 11 2 9 6 0
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ g++ -O1 cache.cpp -o cache
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ ./cache
size : 20
straight: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0
reversed: 19 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
random:   18 13 3 17 15 14 0 8 10 16 2 7 19 5 12 6 4 9 1 11
```

Рис. 1. Проверка корректности обходов массива

Приложение 3. Результаты программы

Таблица 1. Результаты программы

Ticks	KiB	Straight	Reverse	Random
256	1,000	6,26562	6,26562	6,29688
307	1,199	6,21498	6,21498	6,28013
368	1,438	6,27174	6,22826	6,27717
441	1,723	6,20862	6,15873	6,23129
529	2,066	6,24953	6,1966	6,20416
634	2,477	6,19243	6,20505	6,21136
760	2,969	6,21053	6,20789	6,21316
912	3,563	6,1557	6,22807	6,22368
1094	4,273	6,21755	6,21938	6,22486
1312	5,125	6,17988	6,21951	6,18902
1574	6,148	6,21601	6,18043	6,14612
1888	7,375	6,18644	6,22564	6,20975
2265	8,848	6,20044	6,15453	6,19868
2718	10,617	6,18249	6,17881	6,19058
3261	12,738	6,17725	6,30175	6,18399
3913	15,285	6,1656	6,39049	6,2162
4695	18,340	6,19936	6,26368	6,19723
5634	22,008	6,18246	6,6088	6,37167
6760	26,406	6,17751	6,18195	6,17988
8112	31,688	6,19453	6,19773	6,1896
9734	38,023	6,19416	6,19519	8,22046
11680	45,625	6,20479	6,18425	8,43647
14016	54,750	6,19192	6,18279	9,25442
16819	65,699	6,18848	6,23105	10,7541
20182	78,836	6,1983	8,07858	10,4682
24218	94,602	6,29342	6,19969	12,5995
29061	113,520	6,1991	6,19229	11,3162
34873	136,223	6,19476	6,48737	11,6549
41847	163,465	6,3963	6,18883	12,3189
50216	196,156	6,1944	6,19074	13,0484
60259	235,387	6,77831	6,22543	15,3176
72310	282,461	6,20177	6,26038	18,7292
86772	338,953	6,20749	6,20857	22,2091
104126	406,742	6,21582	6,21126	25,0809
124951	488,090	6,21575	6,22836	28,156
149941	585,707	6,21631	6,214	30,8554
179929	702,848	6,21636	6,22942	33,2859
215914	843,414	6,22804	6,21734	35,2454
259096	1012,094	6,21416	6,21599	36,8776
310915	1214,512	6,21641	6,22072	38,3387
373098	1457,414	6,21466	6,24502	44,3035
447717	1748,895	6,24666	6,27191	41,8626
537260	2098,672	6,34379	6,2665	52,1628
644712	2518,406	6,25403	6,39862	70,778
773654	3022,086	6,53752	6,32462	85,9979
928384	3626,500	6,30413	6,32961	98,6443
1114060	4351,797	6,33348	6,39377	117,263
1336872	5222,156	6,38566	6,39482	133,992

1604246	6266,586	6,40017	6,4218	159,196
1925095	7519,902	6,41628	6,41717	175,92
2310114	9023,883	6,44354	6,42719	193,086
2772136	10828,656	6,40592	6,44244	212,799
3326563	12994,387	6,4022	6,44433	223,028
3991875	15593,262	6,4294	6,43534	233,603
4790250	18711,914	6,41996	6,53023	243,019
5748300	22454,297	6,55544	6,42565	248,575
6897960	26945,156	6,42253	6,42073	255,146
8277552	32334,188	6,425	6,42455	261,321

Окончание Табл. 1

Приложение 4. Реальные значения кэша.

```
dasha@dasha-K501UQ:~/masec/evm/lab5/code$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              2
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           78
Model name:                      Intel(R) Core(TM) i3-6100U CPU @ 2.30GHz
Stepping:                        3
CPU MHz:                         2300.000
CPU max MHz:                     2300.0000
CPU min MHz:                     400.0000
BogoMIPS:                        4599.93
Virtualization:                  VT-x
L1d cache:                       64 KiB
L1i cache:                       64 KiB
L2 cache:                        512 KiB
L3 cache:                        3 MiB
NUMA node0 CPU(s):               0-3
```

Рис. 1. Кэш полученный с помощью программы lscpu в Linux.

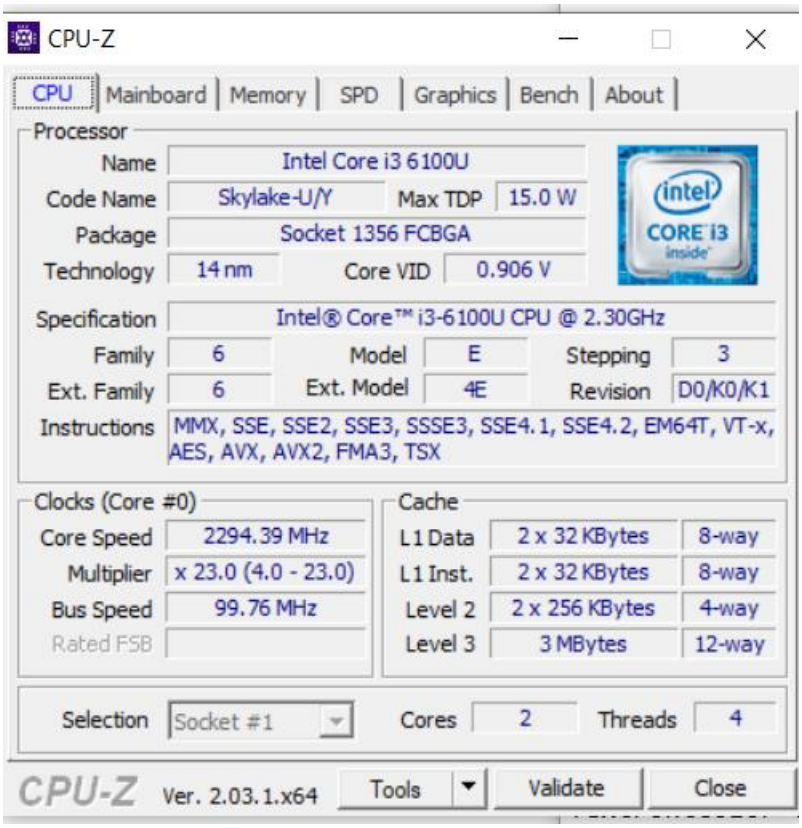

Рис. 2. Кэш полученный с помощью программы CPU-Z в Windows.