Austin Brown

CPE 434-01

3/11/2021

Lab 9

Part 1

Cachegrind will use an independent first level instruction and data cache followed by a unified second level cache for machines with two levels of caches. When you have more than two levels, Cachegrind will simulate the first and last level. The last level has access to main memory, so it has the most impact on performance.

Assignment 1

```
/*
File: test1.cpp
compile as g++ test1.cpp -o test1
*/
using namespace std;
#include <iostream>
main()
{
    int array[1000][1000];
    int i,j;
    for(i=0;i<1000;i++)
        for(j=0;j<1000;j++)
            array[i][j]=0;
    cout << "array[0][0] was " << array[0][0] << endl;
}
```

```
/*
File: test2.cpp
compile as g++ test2.cpp -o test2
*/
using namespace std;
#include <iostream>
main()
{
    int array[1000][1000];
    int i,j;
```

```
    for(i=0;i<1000;i++)
        for(j=0;j<1000;j++)
            array[j][i]=0;
    cout << "array[0][0] was " << array[0][0] << endl;
}
```

L9D1.cpp Output

```
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [11]
└─[$] g++ L9D1.cpp -o a.out
L9D1.cpp:7:6: warning: ISO C++ forbids declaration of ‘main’ with no type [-Wreturn-type]
    7 | main()
      |      ^
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [12]
└─[$] ./a.out
array[0][0] was 0
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [13]
└─[$]
```

L9D2.cpp Output

```
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [16]
└─[$] g++ L9D2.cpp -o a.out
L9D2.cpp:7:6: warning: ISO C++ forbids declaration of ‘main’ with no type [-Wreturn-type]
    7 | main()
      |      ^
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [16]
└─[$] ./a.out
array[0][0] was 0
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/School/cpe434/lab9] - [17]
└─[$]
```

In program 1, array[i][j] is set to 0. In program 2, array[j][i] is set to 0.

Assignment 2

Test 1 Results

```
==396== I   refs:      13,294,042
==396== I1  misses:         1,870
==396== LLi misses:         1,829
==396== I1  miss rate:       0.01%
==396== LLi miss rate:       0.01%
==396==
==396== D   refs:       5,738,224  (4,545,128 rd   + 1,193,096 wr)
==396== D1  misses:        80,392  (   15,378 rd   +    65,014 wr)
==396== LLd misses:        72,883  (    9,518 rd   +    63,365 wr)
==396== D1  miss rate:        1.4% (      0.3%     +       5.4%   )
==396== LLd miss rate:        1.3% (      0.2%     +       5.3%   )
==396==
==396== LL refs:           82,262  (   17,248 rd   +    65,014 wr)
==396== LL misses:         74,712  (   11,347 rd   +    63,365 wr)
==396== LL miss rate:         0.4% (      0.1%     +       5.3%   )
┌─[austinsbrown@DESKTOP-O0AMQ3N] - [/mnt/c/Users/austi/OneDrive/Sch
└─[$]
```

Test 2 Results

```
==398== I   refs:      13,294,042
==398== I1  misses:         1,870
==398== LLi misses:         1,829
==398== I1  miss rate:       0.01%
==398== LLi miss rate:       0.01%
==398==
==398== D   refs:       5,738,224  (4,545,128 rd   + 1,193,096 wr)
==398== D1  misses:     1,017,924  (   15,378 rd   + 1,002,546 wr)
==398== LLd misses:        73,569  (    9,518 rd   +    64,051 wr)
==398== D1  miss rate:       17.7% (      0.3%     +      84.0%   )
==398== LLd miss rate:        1.3% (      0.2%     +       5.4%   )
==398==
==398== LL refs:        1,019,794  (   17,248 rd   + 1,002,546 wr)
==398== LL misses:         75,398  (   11,347 rd   +    64,051 wr)
==398== LL miss rate:         0.4% (      0.1%     +       5.4%   )
```

Test 1 performs far better than test 2 does. You can see that next to label D1 that test 1 has far fewer misses than test 1 does. This is because test 1 finishes working with a row before working on the next column. This allows for more cache hits.

Assignment 3

```c
//file: test 3 compile as test3
#include <stdlib.h>
int main()
{
    char *x = (char*)malloc(100); /* or, in C++, "char *x = new char[
100] */
    return 0;
}
```

The space is never deallocated, resulting in a memory leak.

```c
//file: test4 compile as test4
#include <stdlib.h>
int main()
{
    char *x = (char*)malloc(10);
    x[10] = 'a';
    return 0;
}
```

The maximum index of x is 9. The program is trying to write to memory that it does not have permission to. Also, x is never deallocated.

```c
//file: test5 compile as test5
#include <stdio.h>
int main()
{
    int x;
    if(x == 0)
    {
        printf("X is zero"); /* replace with cout and include iostrea
m for C++*/
    }
    return 0;
}
```

X is never initialized.

Assignment 4

Test 3 Check

```
==588== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==588==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==588==    by 0x10915E: main (in /mnt/c/Users/austi/OneDrive/School/cpe434/lab9/test3)
==588==
==588== LEAK SUMMARY:
==588==    definitely lost: 100 bytes in 1 blocks
==588==    indirectly lost: 0 bytes in 0 blocks
==588==      possibly lost: 0 bytes in 0 blocks
==588==    still reachable: 0 bytes in 0 blocks
==588==         suppressed: 0 bytes in 0 blocks
```

Test 4 Check

```
==589== Invalid write of size 1
==589==    at 0x10916B: main (in /mnt/c/Users/austi/OneDrive/School/cpe434/lab9/test4)
==589==  Address 0x4a4804a is 0 bytes after a block of size 10 alloc'd
==589==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==589==    by 0x10915E: main (in /mnt/c/Users/austi/OneDrive/School/cpe434/lab9/test4)
==589==
==589==
==589== HEAP SUMMARY:
==589==     in use at exit: 10 bytes in 1 blocks
==589==   total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==589==
==589== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==589==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==589==    by 0x10915E: main (in /mnt/c/Users/austi/OneDrive/School/cpe434/lab9/test4)
==589==
==589== LEAK SUMMARY:
==589==    definitely lost: 10 bytes in 1 blocks
==589==    indirectly lost: 0 bytes in 0 blocks
==589==      possibly lost: 0 bytes in 0 blocks
==589==    still reachable: 0 bytes in 0 blocks
==589==         suppressed: 0 bytes in 0 blocks
```

Test 5 Check

```
==590== Conditional jump or move depends on uninitialised value(s)
==590==    at 0x109159: main (in /mnt/c/Users/austi/OneDrive/School/cpe434/lab9/test5)
```

Part 2

Assignment 5

sortingAlgorithims.h

```h
#pragma once

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <math.h>

int * generateArray(long int);
void swap(int *, int *);
void mergeArrays(int *, int, int, int);
void mergeSort(int *, long int, long int);
void insertionSort(int *, long int);
int partition(int *, long int, long int);
void quickSort(int *, long int, long int);
int check(int *, long int);
```

sortingAlgorithims.c

```c
#include "sortingAlgorithims.h"

int *generateArray(long int n)
{
    srand(time(NULL));
    int *arr = (int*)malloc(n * sizeof(int));
    long int i;
    for (i = 0; i < n; i++)
        arr[i] = rand();
    return arr;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
```

```
    *b = temp;
}

void mergeArrays(int *arr, int l, int m, int r)
{
    long int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    //int leftArr[n1], rightArr[n2];
    int *leftArr = (int*)malloc(n1*sizeof(int));
    int *rightArr = (int*)malloc(n2*sizeof(int));

    // Copy data from arr into temp arrays
    for (i = 0; i < n1; i++)
        leftArr[i] = arr[l+i];
    for (j = 0; j < n2; j++)
        rightArr[j] = arr[m+1+j];

    i = 0;
    j = 0;
    k = l;
    // Merge temp arrays back into arr
    while (i < n1 && j < n2)
    {
        if (leftArr[i] <= rightArr[j])
            arr[k++] = leftArr[i++];
        else
            arr[k++] = rightArr[j++];
    }

    // Copy remaining elements of leftArr into arr
    while (i < n1)
        arr[k++] = leftArr[i++];

    // Copy remaining elements of rightArr into arr
    while (j < n2)
        arr[k++] = rightArr[j++];
```

```
    free(leftArr);
    free(rightArr);
}

void mergeSort(int *arr, long int l, long int r)
{
    if (l < r)
    {
        long int m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        mergeArrays(arr, l, m, r);
    }
}

void insertionSort(int *arr, long int n)
{
    long int i, j, key;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int partition(int *arr, long int l, long int h)
{
    int pivot = arr[h];
    long int i = (l - 1);

    long int j;
    for (j = l; j <= h-1; j++)
    {
```

```
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[h]);
    return (i+1);
}

void quickSort(int *arr, long int l, long int h)
{
    if (l < h)
    {
        int pIndex = partition(arr, l, h);
        quickSort(arr, l, pIndex-1);
        quickSort(arr, pIndex+1, h);
    }
}

int check(int *arr, long int n)
{
    long int i;
    int temp = arr[0];
    for (i = 1; i < n; i++)
    {
        if (temp > arr[i])
            return 0;
        temp = arr[i];
    }
    return 1;
}
```

Assignment 6

Sorting Method	Power Consumed (Watts)					
	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Insertion	7.514	8.148	7.572	8.664	7.226	7.825
Merge	5.952	5.321	5.392	5.264	5.569	5.499
Quick	3.846	3.428	4.124	3.727	4.021	3.829

Assignment 7

Sorting Method	Power Consumed (Watts)					
	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Insertion	6.023	6.212	7.178	6.596	6.186	6.439
Merge	5.329	5.432	5.795	5.194	5.193	5.387
Quick	3.233	4.198	3.534	3.134	3.743	3.568

Assignment 8

Insertion sort was by far the worst. The compiler flag did help quite a bit, but it still has the worst complexity of the three. Merge sort was in the middle. It is far more efficient that insertion. Quicksort was the most efficient in terms of energy usage. This is likely because it does not require as much memory. This is because it does not need any immediate arrays. Changing the compiler optimization flag helped all three sorting algorithms.