National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS213: Advanced Programming

Class: BESE – 4B

Lab 3: Cache Optimization

Date: February 21, 2016

Time: 10:00 AM to 12:50 PM

Instructor: Mr. Numair Khan

Lab 3: Cache Optimization

Introduction

We have discussed how the cache is invisible to you as a programmer. However, while you cannot directly control how the cache is used by the processor, you can influence it through the design of your code. Your code should be designed such that cache misses (and, hence, main memory accesses) are minimized. For our third lab, we will be implementing matrix, and vector multiplication through C++ classes designed to optimize memory/cache usage. You will also learn about operator overloading in C++.

Objectives

After performing this lab students will be able to:

- Understand how to lay out and access data structures to maximize cache hits
- Overload operators for user-defined classes
- Learn to use initialization lists in C++
- Use the Linux `perf` command to profile your program

Tools/Software Requirement

- GCC
- Perf profiler tool

Description

A vector can be thought of as an n-element 1D array, and a matrix as a 2D array. Read the following article to refresh your memory of how matrix/vector multiplication works: http://mathinsight.org/matrix_vector_multiplication

Lab Task 1

Your first task is to implement two classes:

1. An n-dimensional vector class
2. An mxn matrix class

Where n may be a very large integer (>10000). The dimensions for both the vector and the matrix should be specified in the constructor of each class. Once an object of the class has been created, the dimensions cannot change. Therefore, it is a good idea to declare n as a constant field in your class. But if n is a constant, how can it be set? To learn how, read the following article on using *initialization lists* in C++:

http://www.cprogramming.com/tutorial/initialization-lists-c++.html

It is up to you to decide how to store the columns and rows of your matrix and vector (in the case of a vector there is only one column) in memory - column, or row major order. Whichever order you choose, make sure your class methods are designed to access the columns and rows in the most optimal manner.

Lab Task 2

For your second task, you are required to implement two methods

1. Multiply an object of the vector class with an object of the matrix class (w = Mv)
2. Multiply two objects of the matrix class (C = AB)

Both methods should be written such that they optimize cache usage.

You are probably aware that not all matrix-matrix, and matrix-vector multiplications are valid. Your code should check the dimensions of the operands to make sure the multiplication operation can proceed. Also, bear in mind that matrix/vector multiplication is not commutative (This means if A and B are matrices, then AB is not always equal to BA).

Hint: The second method (and, in fact, the design of your matrix class) can be greatly simplified if you recall that a single column, or a single row of a mxn matrix is nothing more than an n, or m-dimensional vector, respectively.

Lab Task 3

Overload the * operator. Read the following article to learn about operator overloading in C++:

http://www.cprogramming.com/tutorial/operator_overloading.html

If `M` and `Q` are matrices, and `v` a vector, then the following operations should be allowed by your code

```
1. M * n //vector – matrix multiplication
2. M * Q //matrix – matrix multiplication
```

Lab Task 4

Now that you have your classes in place, it's time to test how well your code utilizes the cache.

1. Create a nxn matrix and an n element vector, where n = 1000
2. Initialize both to random values (by now, you're probably an expert on which random number generator works best).
3. Multiply the vector and the matrix: `w = M * v`
4. Compile your code on a Linux machine: `gcc mult.c -o mult`
5. Run your code using the `perf` profiler: `perf stat ./mult` and count the number of cache misses. The cache misses is the metric your code should optimize.

Your code will be graded by multiplying a matrix and vector (or two matrices) with n equal to some random number, and seeing how many cache misses occur. The fewer the cache misses, the better.

Deliverables

1. Your source code