263-2300-00: How To Write Fast Numerical Code

Assignment 3

Due Date: Thu March 31 17:00

Maximal points: 200+10

http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

This is a 2 week homework and hence counts twice as much as the previous homeworks.

Submission instructions: Your submission for this assignment will include two parts. Submit both parts into your SVN directory into the according folder hw03. Keep in mind that we check if the code was copied from other people, so don't do it!

Part 1: Plots and answers: The first part will be a file that contains the required plots and answers to the questions. If you use other programs (such as MS-Word) to create your assignment, convert them to PDF (google for 'pdfcreator' for a free conversion program).

Part 2: Source code: The second part will consist of your source code. Follow the instructions below to create, name, and submit your source code. For all questions where you are asked to provide C code, we provide a corresponding C template file that you need to use to answer the question. In particular,

- do NOT change the signature of the functions
- do NOT change the type of global arguments
- comply with the given environment variables, do not add others
- do not cross-reference functions among your submitted files. Each .c file should be completely independent should be compilable on its own, (when compiled with our own main.c that you don't have access to).
- clean up your code as much as possible, do not leave in debug statements
- we provide a helpful sample main.c to show you how we would like the code to be structured. You are free to use it or not use it. It doesn't come with a timer, as you need to implement one. (possibly using the one from Assignment 2). Do not submit this sample file.

Verifying your code: All code that you produce as a part of this assignment (and future assignments too!) needs to be verified for computational correctness. For MMM, the easiest way is to compare to the standard triple loop (from assignment 2) for a few randomly selected input matrices. We will independently verify your code for correctness. Incorrect programs will not receive any credit.

Submitting your code: Your C files corresponding to the questions should be named code0.c, code1.c, code2.c, and code3.c. In all, you will submit the 4 code<n>.c files, and the finalcode.c file, plus any extra credit files. Do NOT change file names! Do not zip or otherwise archive the files.

1. *Miss rate (40 pts)* We consider the following program:

```
typedef double matrix[2][8]
double comp(matrix A) {
        int i;
        double t = 1000.0;
        for (i = 0; i < 7; i++) { // note: 7 not 8 because of the boundary
                t = t/(A[0][i] + A[0][i+1]);
                t = t/(A[1][i] + A[1][i+1]);
        }
        return t;
}
```

We assume that a double requires 8 bytes, and that the array is cache aligned (that is A[0][0] is mapped to the first set and the first position in a cache block). Further, A has been initialized to contain only positive numbers. We assume a cold cache and ignore i and t in the cache analysis (they are held in registers). Recall that the miss rate is defined as $\frac{\#misses}{\#accesses}$. (Hint: It helps to draw cache and array.)

(a) How many times is A accessed in this program?

(b) The cache is direct mapped, has a size of 64 bytes, and 4 sets.

   i. How many doubles fit into one cache block?

   ii. What is the miss rate of the above program?

(c) The cache is 2-way set associative, has a size of 128 bytes, and 4 sets.

   i. How many doubles fit into one cache block?

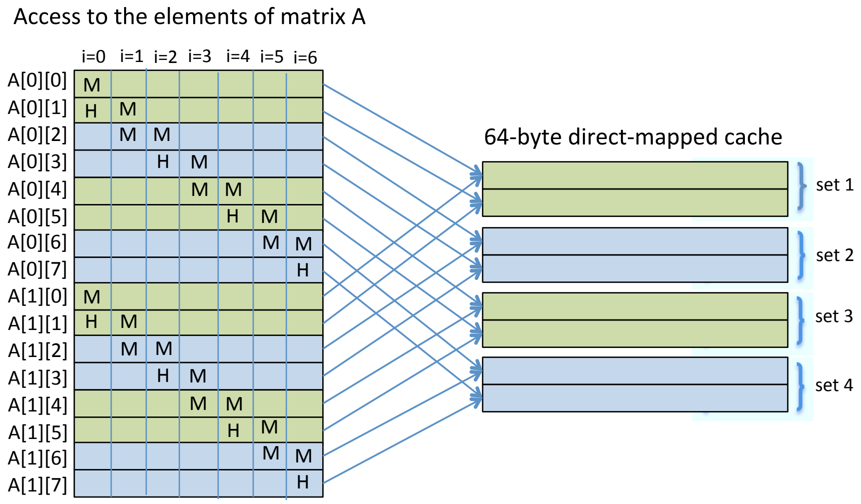
   ii. What is the miss rate of the above program?

Solution:

(a) On each iteration of the loop, 4 elements of A are accessed. As the loop is executed 7 times, the matrix is accessed 28 times.

(b) i. The cache can store 64 bytes or 8 doubles. Since it is a direct mapped cache, the 64 bytes are divided into 4 blocks or sets. Thus, each block contains 16 bytes or 2 doubles.

   ii. Iteration 1: A[0][0] is loaded into cache, and in the same block A[0][1] is loaded. Thus, access to A[0][0] is a miss and access to A[0][1] is a hit. Then, A[1][0] is loaded into cache, and in the same block A[1][1] is loaded. Thus, access to A[1][0] is a miss and access to A[1][1] is a hit. Iteration 2: A[0][1] is accessed again and it is not in cache any more, since A[1][0] and A[1][1] were stored previously in the same block. Thus, A[0][1] is a miss. A[0][2] is loaded in a different block and it is accessed for the first time. Thus, access to A[0][2] is a miss. The same pattern is repeated as shown in the figure. Therefore, miss rate equals 20/28.

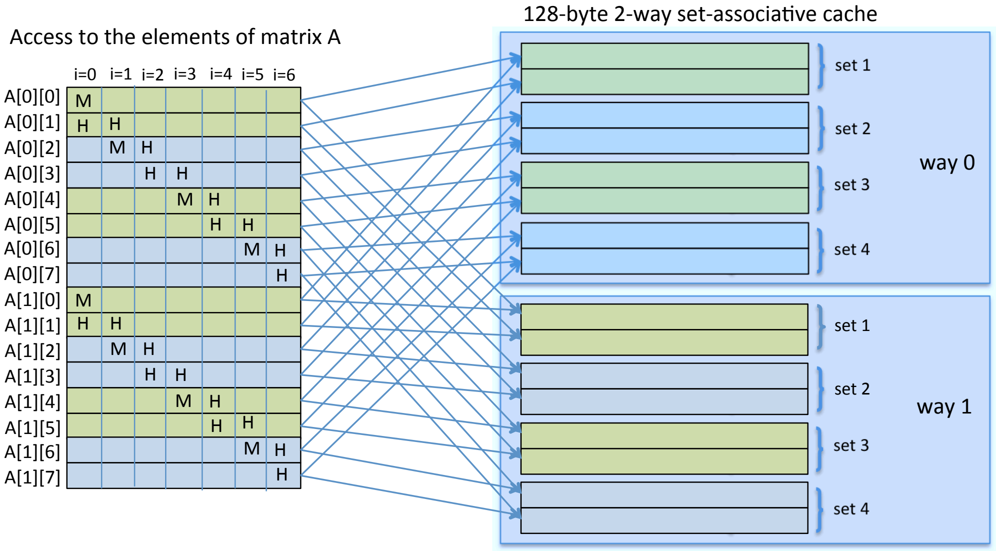


Access to the elements of matrix A

	i=0	i=1	i=2	i=3	i=4	i=5	i=6
A[0][0]	M						
A[0][1]	H	M					
A[0][2]		M	M				
A[0][3]			H	M			
A[0][4]				M	M		
A[0][5]					H	M	
A[0][6]						M	M
A[0][7]							H
A[1][0]	M						
A[1][1]	H	M					
A[1][2]		M	M				
A[1][3]			H	M			
A[1][4]				M	M		
A[1][5]					H	M	
A[1][6]						M	M
A[1][7]							H

(c) i. The cache can store 128 bytes or 16 doubles. Since it is a 2-way set associative cache, the 128 bytes are divided into 4 sets. Thus, each set contains 32 bytes or 4 doubles. In turn, each set is divided in 2 ways. Therefore, each block contains 16 bytes or 2 doubles.

ii. There are two blocks in cache where each element can be stored in cache. Thus, the displacement that happened for the direct mapped cache, when the second row of the matrix, does not happen in this case. As shown in the picture, miss rate equals 8/28.



Access to the elements of matrix A

	i=0	i=1	i=2	i=3	i=4	i=5	i=6
A[0][0]	M						
A[0][1]	H	H					
A[0][2]		M	H				
A[0][3]			H	H			
A[0][4]				M	H		
A[0][5]					H	H	
A[0][6]						M	H
A[0][7]							H
A[1][0]	M						
A[1][1]	H	H					
A[1][2]		M	H				
A[1][3]			H	H			
A[1][4]				M	H		
A[1][5]					H	H	
A[1][6]						M	H
A[1][7]							H

2. *Reuse (20 pts)* Compute the reuse (as defined in class), both exact and in O-notation, for the following. Give enough detail so we know how you did it.

(a) matrix-vector multiplication (MVM) in the form $y = Ax + y$, where x, y are vectors of length n and A is $n \times n$.

(b) the scalar or dot product $\alpha = x * y = \sum_{i=0}^{n-1} x_i y_i$, where x, y are of length n.

Solution:

(a) MVM of form $y = Ax + y$ requires $\underbrace{n^2}_{multiplications} + \underbrace{n^2}_{additions}$. Therefore the reuse is

$$\begin{aligned} Reuse &= \frac{\#\text{operations}}{\text{size(input)} + \text{size(output)}} \\ &= \frac{2n^2}{n^2 + 2n} \\ &= O(1) \end{aligned}$$

(b) In case of the scalar dot product case we have $\underbrace{n}_{multiplications} + \underbrace{n-1}_{additions}$ operations.

$$\begin{aligned} Reuse &= \frac{2n-1}{2n+1} \\ &= O(1) \end{aligned}$$

3. *Blocking analysis for MVM (40 pts)* We consider double precision MVM in the form $y = Ax$, where x, y are of length n and A is $n \times n$. The goal is to perform a cache miss analysis with and without blocking analogous to the one done for MMM in class.

   Assume a cache size of C doubles and that C is much smaller than n. Assume a cache block size of 8 doubles and a cold (empty) cache.

   (a) With respect to locality, what is the fundamental difference between MMM and MVM?

   (b) Using the above assumptions, compute the number of cache misses of a standard double loop implementation based on

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
      y[i] += A[i][j]*x[j];
```

   Give enough detail so we know how you did it.

   (c) Now assume a blocked implementation based on dividing A into blocks of size $b \times b$, where 8 divides b and b divides n:

```
for (i = 0; i < n; i+=b)
   for (j = 0; j < n; j+=b)
      for (k = i; k < i+b; k++)
         for (l = j; l < j+b; l++)
            y[k] += A[k][l]*x[l];
```

   Compute the number of cache misses as a function of b. Give enough detail so we know how you did it.

   (d) Continuing the previous question, derive the largest block size such that one block and two corresponding chunks of x and y fit into the cache (of size C). Show your work. What is the reduction in the number of cache misses for this block size?

Solution: A complete solution analogous to how we did it in class (visual) is appended in the end.

(a) MVM has a resuse of O(1) compared to the reuse O(n) of MMM

(b) In class (MMM) we did not analyze the number of misses for the output. Here the output is `y`. Including or excluding these misses is both correct as solution. Below we mark the part of the cache misses that is due to `y` with parentheses.
Within one iteration of the j loop we get $\frac{2n}{\text{cache block size}}$ misses

$$y[i] + = \underbrace{A[i][j]}_{\frac{n}{8}} * \underbrace{x[j]}_{\frac{n}{8}};$$

Within the i loop we get $\frac{n}{\text{cache block size}}$ misses fory:

$$\underbrace{y[k]}_{\frac{n}{8}} += A[k][l] * x[l];$$

The j loop is iterated over n times, therefore yielding

$$\frac{2n}{8} * n$$

misses. The total number of misses is therefore

$$\frac{n^2}{4} + \left(\frac{n}{8}\right)$$

(c) For the blocked analysis we assume that $2b + b^2 \leq C$ and that the cache is fully associative like we also assumed in the class. We start off by looking at the matrix $A[k][l]$. We walk from j to $j + b$ within every l loop resulting in $\frac{b}{8}$ misses. Looking at the k index of $A[k][l]$ we see that we have the same scenario again for every iteration of the k loop. So all together it is

$$\underbrace{\frac{b}{8}}_{\text{misses}} * \underbrace{b}_{\text{\#k loop}} * \underbrace{\frac{n}{b}}_{\text{\#j loop}} * \underbrace{\frac{n}{b}}_{\text{\#i loop}}$$

misses for $A[k][l]$. Looking at $x[l]$ we see that we also get $\frac{b}{8}$ misses within every l loop, but it is already in cache within every iteration of the k loop since it does not depend on it. Therefore the total number of misses for $x[l]$ is

$$\underbrace{\frac{b}{8}}_{\text{misses}} * \underbrace{\frac{n}{b}}_{\text{\#j loop}} * \underbrace{\frac{n}{b}}_{\text{\#i loop}}$$

Finally when we look into the expression $y[k]$ we see that it only depends on k - so it yields $\frac{b}{8}$ misses per k loop. Since it does not depend on j it will be in cache for every j and only yield more cache misses during the i loop resulting in

$$\underbrace{\frac{b}{8}}_{\text{misses}} * \underbrace{\frac{n}{b}}_{\text{\#i loop}}$$

To sum up we get a total of

$$\underbrace{\left(\frac{n}{8}\right)}_{y[k]} + \underbrace{\frac{n^2}{8}}_{A[k][l]} + \underbrace{\frac{n^2}{8b}}_{x[l]}$$

misses.

(d) Deriving the cache misses depending on C for the largest possible block size is quite straight forward knowing that the largest possible C follows the already mentioned formular $2b + b^2 \leq C$ which gives us

$$b = \sqrt{C+1} - 1,$$

or, more precisely,

$$b = \lfloor \sqrt{C+1} - 1 \rfloor.$$

Plug in this into our result from before we get a miss rate of

$$\underbrace{(\frac{n}{8})}_{y[k]} + \underbrace{\frac{n^2}{8}}_{A[k][l]} + \underbrace{\frac{n^2}{8 * (\sqrt{C+1} - 1)}}_{x[l]}$$

misses.

The reduction in cache misses is best computer as quotient but the difference is also fine since it was not very clearly phrased.

Quotient (excluding the term for y for simplicity):

$$\frac{2}{1+\frac{1}{\sqrt{C+1}-1}} \approx 2.$$

Note the difference to MMM where it was about $2.5\sqrt{C}$.

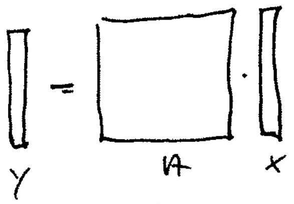
Difference:

$$\frac{n^2}{8}\left(1-\frac{1}{\sqrt{C+1}-1}\right).$$

Solution Q 5:

a.) MVM has $O(1)$ reuse, MMM has $O(n)$

b.)



assumptions: $n \gg$ cache size C
cache line: 8 doubles

triple loop:

1. element of y: $\underbrace{\frac{n}{8}}_{\text{row of } A} + \underbrace{\frac{n}{8}}_{x} = \frac{n}{4}$

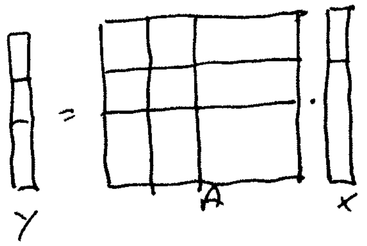
2. element of y: same

. . . . .

sum: $n \cdot \frac{n}{4} = \frac{n^2}{4}$ cache misses

[with y: $\frac{n^2}{4} + \frac{n}{8}$ (compulsory misses for y)]

c.) blocked: $8|b$, $b|n$



1. segment of y: $\underbrace{\frac{b^2}{8}}_{1 \text{ block}} \cdot \underbrace{\frac{n}{b}}_{\text{no. of blocks}} + \underbrace{\frac{n}{8}}_{x} = \frac{nb}{8} + \frac{n}{8}$

2. segment of y: same

. . . . .

sum: $\frac{n}{b}\left(\frac{nb}{8} + \frac{n}{8}\right) = \left(\frac{1}{8} + \frac{1}{8b}\right) n^2$

provided: $\underbrace{b^2}_{\text{1 block of } A} + \underbrace{2b}_{\text{1 segment of } x, y} \le C$

[with y: $\left(\frac{1}{8} + \frac{1}{8b}\right) n^2 + \frac{n}{8}$]

d.) largest b such that $b^2 + 2b \le C$?

$b^2 + 2b - C = 0 \iff b_{1/2} = -1 \pm \sqrt{1+C} \implies b_{max} = \lfloor -1 + \sqrt{C+1} \rfloor$

cache misses: $\left(\frac{1}{8} + \frac{1}{\lfloor 8(-1+\sqrt{C+1}) \rfloor}\right) n^2$

reduction (! form the quotient to get the x-fold improvement):

$$\frac{\frac{n^2}{4}}{\left(\frac{1}{8} + \frac{1}{8(-1+\sqrt{C+1})}\right) n^2} = \frac{2}{1 + \frac{1}{-1+\sqrt{C+1}}} = \frac{2(-1+\sqrt{C+1})}{\sqrt{C+1}} \simeq 2$$

very different to gain for MMM ($\simeq 2.5\sqrt{C}$)