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

Assignment 3: 100 points
Due Date: Thu March 20 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring14/course.html
Questions: fastcode@lists.inf.ethz.ch

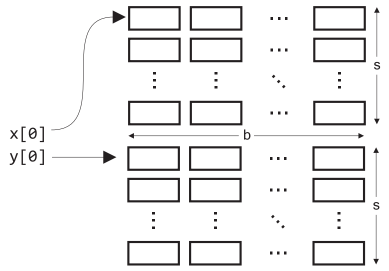
Submission instructions (read carefully):

- (Submission)
  We set up a SVN Directory for everybody in the course. The Url of your SVN Directory is https://svn.inf.ethz.ch/svn/pueschel/students/trunk/s14-fastcode/YOUR.NETZH.LOGIN/ You should see subdirectory for each homework.
- (Late policy)
  You have 3 late days, but can use at most 2 on one homework. Late submissions have to be emailed to fastcode@lists.inf.ethz.ch.
- (Formats)
  If you use programs (such as MS-Word or Latex) to create your assignment, convert them to PDF and submit to svn in the top level of the respective homework directory. Call it homework.pdf.
- (Plots)
  For plots/benchmarks, be concise, but provide necessary information (e.g., compiler and flags) and always briefly discuss the plot and draw conclusions. Follow (at least to a reasonable extent) the small guide to making plots (lecture 5).
- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. *Cache mechanics (30 pts)* We consider a direct-mapped cache with parameters $(S, E, B) = (2s, 1, 8b)$ and the code bellow, having constant parameter $t < s$:

```
double x[b*s + t], y[b*s + t];
double sum = 0; int i;
for (i = 1; i < 2(b*s + t); i++)
  sum += y[i%(b*s+t)] * x[b*i%(b*s+t)];
```



We assume that `x[0]` goes into the first slot of the first block, `y[0]` goes into the first slot of the $(s+1)^{\text{th}}$ block, and that `sum` and `i` are held in registers (meaning you do not need to consider them in the cache analysis) and b, s and t are predefined constants.

(a) Determine the miss rate for array `y`, and express it as a function of the parameters s, b and t.

(b) Determine the miss/hit sequences for x and y (something like x: `MHHMHHM..`) for $(s, b, t) = (6, 2, 1)$.

(c) What is the operational intensity of the entire code assuming $(s, b, t) = (6, 2, 1)$?

Solution.

(a) While general assumption in a direct-mapped cache is that memory mappings of arrays can overlap in the cache, the analysis of conflicts can lead to severe complications in this problem. To analyze the miss rates of the arrays, we start with a non-overlapping version of the cache, assuming that `x` resides in the first s blocks and `y` resides in the second set of s blocks. We then extend the solution to the general overlapping case, where the last t values of `x` are mapped into the first $\lceil t/b \rceil$ set of blocks of `y` and the last t values of `y` are mapped into the first $\lceil t/b \rceil$ set of blocks of `x`.

Non-overlapping cache. To solve this problem we use an auxiliary function:

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

To analyze the miss rates of the array `y`, we only need to consider cache misses. Lets assume that $p = floor(t/b)$ and $q = p \mod b$. We can now illustrate the array accesses as defined in the loop, into 5 meaningful regions:

Region	Array Access	y-Range	Cache Line Range
1	`y[1%(b*s+t)] until y[b*s%(b*s+t)]`	$[1, b \cdot s)$	$s + [floor(1/b), s)$
2	`y[b*s%(b*s+t)] until y[(b*s+t)%(b*s+t)]`	$[b \cdot s, b \cdot s + t)$	$s + [0, p + f(q))$
3	`y[(b*s+t)%(b*s+t)] until y[(b*s+2*t)%(b*s+t)]`	$[0, t)$	$s + [0, p + f(q))$
4	`y[(b*s+2*t)%(b*s+t)] until y[(2*b*s+t)%(b*s+t)]`	$[t, b \cdot s)$	$s + [p + f(q), s)$
5	`y[(2*b*s+t)%(b*s+t)] until y[(2*b*s+2*t)%(b*s+t)]`	$[b \cdot s, b \cdot s + t)$	$s + [0, p + f(q))$

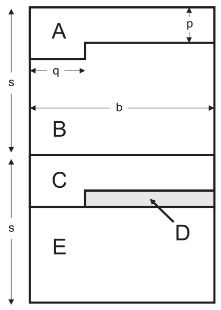
Now we can calculate the cache misses per region:

i. Region 1. has $s - floor(1/b)$ mandatory misses (since we start with `i=1` in the loop).
ii. Region 2. has $p + f(q)$ misses (we load the remaining t values)
iii. Region 3. has $p + f(q)$ misses. Except when $t = 0$ and $b = 1$. In this case `y[0]` gets loaded for the first time and creates cache miss. So there are additional $(1 - f(t)) \cdot floor(1/b)$ misses.
iv. Region 4. has no misses. All elements have already been loaded into the cache in Region 1.
v. Region 5. has $p + f(q)$ misses.

Therefore total misses $s + 3(p + f(q)) - f(t) \cdot floor(1/b)$. Finally the miss-rate is:

$$M(s, b, t) = \frac{s + 3\left[floor(t/b) + f(t \bmod b)\right] - f(t) \cdot floor(1/b)}{2(b \cdot s + t) - 1} \tag{1}$$

General overlapping case. Now, we need to take conflicts with array `x` into considerations. We illustrate the cache in the figure bellow, and explain the accesses as they occur. Note that `x` occupies regions A, B and C, and `y` occupies regions C, D, E and A. Therefore, conflicts can only occur while accessing regions A, C and D. Array access goes as follows:



i. We traverse the `y` array going through regions C, D, E, and A, twice.

ii. While at region C, there are no conflicts with `x` since $t < s$, and we traverse `x` in the A and B regions.

iii. While at region D, there might be conflicts with `x`

iv. There are no conflicts with `x` in region E, since `x` occupies A, B, and C.

v. While at region A, there are no conflicts with `x`, since $t < s$

When accessing array `x`, we jump through cache lines. Therefore in $b \cdot s + t$ iterations, we will jump in every line that has been occupied by `x`. Therefore, once we re-start iterating through `y` array, regions A, C and D will be overwritten with values of `x`. Region E will not be modified, so we will experience no misses when going through it again. Therefore, the minimum number of misses that occur in this case, ignoring the conflicts, is equivalent to the non-overlapping cache version, defined in 1. However, conflucts that occur in region D, introduce additional misses, and we need to include them in the final result.

Region D is accessed when $q > 0$ for values of $i \in \{t, \dots, t+b-q-1\}$. In order for a conflict to happen, we must access value of `x` that lands in the p-th block of C region in the cache, before we access one of $y[i]$ values. Since `y` gets accessed before `x`, we only need to look into $i \in \{t, \dots, t+b-q-2\}$. In order to have conflict, we must have at least one i and j such that:

$$bi \bmod (bs+t) = bs + bp + j \text{ for } j \in \{0, \dots, q-1\} \text{ and } i \in \{t, \dots, t+b-q-2\} \tag{2}$$

or using the congruence relation and its properties:

$$\begin{aligned} bi \equiv{}& \; bs + bp + j \pmod{bs+t} \\ bi \equiv{}& \; bp + j - t \pmod{bs+t} \\ bi \equiv{}& \; j - q \pmod{bs+t} \end{aligned} \tag{3}$$

Therefore, the number of conflicts is the number of m values that satisfy the following equation:

$$bi = m(bs+t) + j - q \text{ for } j \in \{0, \dots, q-1\} \text{ and } i \in \{t, \dots, t+b-q-2\} \tag{4}$$

Getting the exact number of m values that satisfies 4 is a task which is harder then anticipated. An upper bound of m can easily be estimated using the ranges of f and i:

$$m \le \left\lfloor \frac{b * i_{max} + g - j_{min}}{bs+t} \right\rfloor = \left\lfloor \frac{b*(t+b-q-2)+q}{bs+t} \right\rfloor \tag{5}$$

Since we traverse region D two times, we will experience the same conflicts twice. We also have to note that conflicts in D will not occur at all if $q = 0$. Each of the conflicts will introduce exactly one miss in the `y` array. Finally:

$$M(s,b,t) \le M_{rate}(s,b,t) \le M(s,b,t) + 2 \cdot f(q) \frac{\left\lfloor \frac{b*(t+b-q-2)+q}{bs+t} \right\rfloor}{2(bs+t)-1} \tag{6}$$

We also provide a Scala validation code for our bound calculations.

(b) `x: MMMM MMMH HHHHM HHHH HMHH HHHH`
`y: MMHM HMHM HMHMM HHHH HHHH HHHM`

(c) Misses for both `x` and `y`: 9. We load 16 bytes on each miss. Therefore $Q = \frac{2*2*(6*2+1)-2}{(9+9)*16} = \frac{50}{288}$

2. *Cache mechanics (35 pts)* Consider the following double loop computation:

```
void mvm(double A[3][4], double x[4], double y[3]) {
  for (i = 0; i < 3; i++)
    for (j = 0; j < 4; j++)
      y[i] += A[i][j]*x[j];
}
```

(a) Determine the upper bound I for the operational intensity considering only read traffic.

The execution is performed using a write-back/write-allocate cache with $(S, E, B) = (2, 2, 8)$, where S is the number of sets, E is the associativity, and B the block size in bytes. Assume there is no aliasing between the three arguments, that matrix A is stored row-wise, and that the cache is cold and uses LRU replacement. Also, assume that $A[0][0]$, $x[0]$, and $y[0]$ map to the first set of the cache, and that variables i and j are held in registers.

(b) Compute the miss rate M_{L1}.

(c) Compute the operational intensity I_{L1} (together with reads include also writes due to eviction).

Now assume the presence of a write-back/write-allocate L2 cache with parameters $(S, E, B) = (8, 2, 16)$. Assume that A[0][0] maps to the first set of L2, while $x[0]$ and $y[0]$ to the second-last and the last set respectively. The two caches are cold and use LRU replacement.

(d) Compute the miss rate M_{L2}.

(e) Compute the operational intensity I_{L2} (together with reads include also writes due to eviction).

Provide enough details about your reasoning.

Solution:
See handwritten notes.

3. *Associativity (30 pts)* We consider a direct mapped cache given by $(S, E, B) = (S, 1, B)$, where S is the number of sets and B the block size in bytes. To reduce the number of cache misses we design a second cache given by (S', E', B') with $E' = 2$, $B' = B$, and $S' = S/2$. In words, this cache has the same size and block size but higher associativity. We assume LRU replacement.

   We claim that for any code the number of cache misses on the second cache is equal or smaller than the number of misses on the first cache. Do you agree with this statement? If yes, provide a proof. If no, provide a counterexample (i.e., sketch a function operating on some array where it does not hold).

   Solution:

   The statement is false. Below we provide a simple counterexample where the direct mapped cache produces less misses than the 2-way associative cache:

```
char x[2*B*S];

char t = x[0];
t += x[B*S/2];
t += x[3*B*S/2];
t += x[0];
```

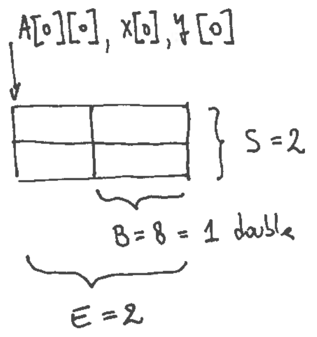
   Assume that x maps to the first set of both caches and excluding variable t from the analysis, we obtain the following hit/miss sequences for the two cases:

   Direct mapped: M M M H

   2-way associative: M M M M

E2) a) $I = \frac{24}{(12+7)8} \simeq 0.16 \frac{f}{B}$

b)
c) L1:



d) L2:
e)

$E = 2$, $S = 8$

A[0][0]	A[0][1]	Y[2]	—
A[0][2]	A[0][3]		
A[1][0]	A[1][1]		
A[1][2]	A[1][3]		
A[2][0]	A[2][1]		
A[2][2]	A[2][3]		
X[0]	X[1]		
Y[0]	Y[1]	X[2]	X[3]

B=16= 2 doubles

$\xrightarrow{j}$

	A	M	M	M	M
i=0	X	M	M	$\textcircled{M}$	M
	Y	M	H	M	H
	A	M	M	M	M
i=1	X	$\textcircled{M}$	$\textcircled{M}$	M	$\textcircled{M}$
	Y	M	M	H	M
	A	M	M	M	M
i=2	X	M	$\textcircled{M}$	$\textcircled{M}$	M
	Y	M	H	M	H

write back to y

$MR_{L1} = \frac{31}{36} = 86\%$

$I_{L1} = \frac{24}{(\underset{\uparrow \atop R}{31} + \underset{\uparrow \atop WB}{6})8} \simeq 0.08 \frac{f}{B}$

	A	M	H	M	H
i=0	X	M	H	M	H
	Y	M		H	
	A	M	H	M	H
i=1	X	H	H	H	H
	Y	H	H		H
	A	M	H	M	H
i=2	X	H	H	H	H
	Y	M		H	

$MR_{L2} = \frac{10}{31} = 32\%$

$I_{L2} = \frac{24}{20.8} = 0.15 \frac{f}{B}$