263-2300-00: How To Write Fast Numerical Code
Assignment 3: 100 points
Due Date: Thu March 22 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring12/course.html
Questions: fastcode@lists.inf.ethz.ch

Exercises:

1. *Cache mechanics (20 pts)* We consider a cache with parameters $(S, E, B) = (2, 1, 16)$ and the following code:

```
double x[5];
double sum = 0; int i;

for (i = 0; i < 10; i++)
  sum += x(3*i % 5);
```

We assume `x` is cache-aligned (meaning `x[0]` goes into the first position of the first block) and that `sum, i` are held in registers (meaning you do not need to consider them in the cache analysis).

(a) How many doubles fit into one cache block?
(b) Determine the miss/hit sequence (something like MHHHMHHM..).
(c) Determine the miss rate.

It helps to draw the cache. Provide enough detail so we see how you did it.

Solution: See handwritten part.

2. *Cache mechanics (25 pts)* The heart of the game SimAquarium is a tight loop that calculates the average position of 256 algae. You are evaluating its cache performance on a machine with a 1024-byte direct-mapped data cache with 16-byte blocks ($B = 16$). You are given the following definitions:

```
struct algae_position {
  float x;
  float y;
};

struct algae_position grid[16][16];
float total_x = 0, total_y = 0;
int i,j;
```

You should also assume the following:

- `sizeof(float) == 4`.
- `grid` begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the grid array `grid`. the variables `i, j, total_x, total_y` are stored in registers.

We consider the following code:

```
for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++) {
    total_x += grid[j][i].x;
    total_y += grid[j][i].y;
  }
}
```

Analyze the cache performance of this code (provide some short explanations so we see how you got the result; it helps to draw the cache):

(a) What is the total number of reads?

(b) What is the total number of reads that miss in the cache?

(c) What is the miss rate?

(d) What would the miss rate be if the cache were twice as big?

Solution: Based on the given assumptions the cache can hold only half of the working set.

(a) $16 \cdot 16 \cdot 2 = 512$.

(b) 256. Accessing the x-coordinate of a grid point is always a miss, while accessing its y-coordinate is always a hit.

(c) $\frac{256}{512} = 0.5$.

(d) $\frac{128}{512} = 0.25$. Having a cache twice as large would allow for storing the entire grid. The only misses would the be cold ones.

3. *Write back/write allocate caches (20 pts)* We consider the execution of the following loop

```
// x, y, z are vectors of length n (data type double)
for (i = 0; i < n; i++)
  z[i] = x[i] + y[i];
```

We assume a system with one cache that is write back/write allocate and a cache size of C doubles ($8C$ bytes). The loop above is executed with cold cache. Derive a formula, based on n and C, that estimates the number of bytes transferred between memory and the cache during the execution of the loop. Do not consider possible conflicts between the placement of the vectors x, y, z in cache. Provide enough detail so we understand how you got the result.

Solution: See handwritten part.

4. *MMM analysis (20 pts)* In class we estimated the number of cache misses for a triple loop matrix-matrix multiplication (MMM):

```
// A, B, C, are n x n matrices (data type double)
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    C[i][j] = 0;
    for (k = 0; k < n; k++)
      C[i][j] = C[i][j] + A[i][k]*B[k][j];
```

We assume a cache with LRU replacement and a size of C doubles, a cache block size of 8 doubles, and that n is much larger than C and obtained $\frac{9}{8}n^3$ misses.

(a) Estimate the number of misses of the triple loop in the more realistic scenario $16n \leq C$. You can ignore the misses incurred by accessing the array `C`.

(b) Based on this result give an upper bound on the operational intensity $I(n)$ as a function of n.

Provide enough detail so we understand how you got the result.

5. *Fully associative is always best? (10 pts)* A fully associative cache imposes the least restrictions on the placements of blocks transferred from memory. This lead to the following question: does a fully associative cache always produce less or the same number of cache misses than a not fully-associative cache of the same size and with the same block size (we assume LRU replacement)? Formally, the fully associative cache is described by $(S, E, B) = (1, E, B)$ and the not fully associative one by (S', E', B) with $S'E' = E$.

   If this is true, proof it. Otherwise find a counter example (i.e., an array access pattern where it does not hold).

Solution: See handwritten part.

Solution HW 3

1.)

a.) $B = 16 \Rightarrow$ 2 doubles

b.) The sequence is
M M H M H M H H M H (5 misses, 5 hits)

c.) $\frac{5}{10} = \frac{1}{2}$

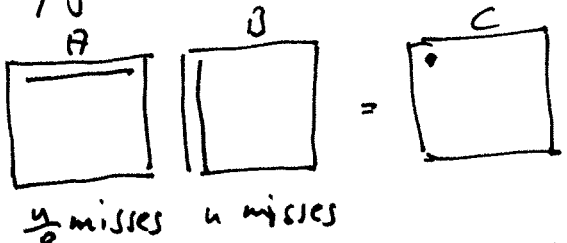
3.) There will certainly be $3n$ loads into cache. (Since z is loaded because of write-allocate)

For $3n \leq C$, that's all.

If $3n > C$, we can assume that the cache holds $\frac{C}{3}$ elements of each x, y, z. Hence, $n - \frac{C}{3}$ elements of z are written back.

Result: $\approx 3n + \max\{0, n - \frac{C}{3}\}$
(one could place floor or ceiling around $\frac{C}{3}$)

4.) $i = 0, j = 0$:



$\frac{n}{8}$ misses n misses

now: 1. row of A and 1.-8. column of B in cache

$\Rightarrow$ no misses for $j = 1 \ldots 7$

$j = 8$: n misses

$j = 9 \ldots 15$: no misses

etc.

Total for $i = 0$: $\frac{n}{8} + \frac{n^2}{8}$ (A, B)

a.) Same for all $i \Rightarrow$ Total $= \frac{1}{8}(n^3 + n^2)$

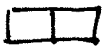
b.) $I(n) \leq \frac{2n^3}{n^3 + n^2} \frac{\text{flops}}{\text{byte}} \approx 2 \frac{\text{flops}}{\text{byte}}$

5.) Surprisingly, fully associative is not always best.

Example:

a.) fully associative cache $(S, E, B) = (1, 2, 8)$ (8 ↑ 1 double)

b.) non-fully ass. cache $(S', E', B) = (2, 1, 8)$

cache a.): 

cache b.): 

Assume an array x of doubles s.t. x[0] is mapped

Consider the access pattern

0 1 3 0 5 7 0 1 3 0 5 7 ....

On cache a.): all misses

On cache b.): M M M H M M H M M H M M ....