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

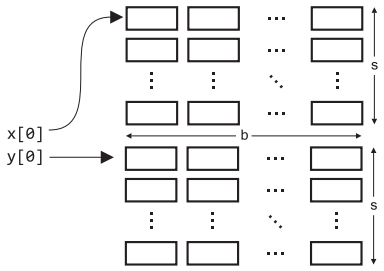
Submission instructions (read carefully):

- (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 $(s, b, t) = (6, 2, 1)$?

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.

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).