263-2300-00: How To Write Fast Numerical Code
Assignment 2: 100 points
Due Date: Thu March 14 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring13/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/s13-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. *Short project info (10 pts)* Submit the following about your project (be brief):
   (a) An exact (as much as possible) but also short, problem specification for your class project.
   For example for MMM, one can be very precise, and it could be like this:
   Our goal is to implement matrix-matrix multiplication specified as follows:
   Input: Two real matrices A, B of compatible size, $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{k \times m}$. We may impose divisibility conditions on n, k, m depending on the actual implementation. *Output:* The matrix product $C = AB \in \mathbb{R}^{n \times m}$.
   (b) The algorithm you plan to consider for the problem. You can actually sketch the algorithm or just give the name and a precise reference (e.g., a link to a publication plus the page number) that explains it.
   (c) A very short explanation of what kind of code already exists and in which language it is written.
2. *Microbenchmarks: Mathematical functions (25 pts)* Since people often ask how expensive sin, cos, etc. are, let's figure it out. Determine the runtime (in cycles) of the following computations (x, y are doubles) as accurately as possible. You can use this template :
   (a) $y = \sin x$
   (b) $y = \log(x + 0.1)$
   (c) $y = e^x$
   (d) $y = \frac{1}{x+1}$
   (e) $y = x^2$

   Initalize x to the values 0, 0.9, 1.1 and 4.12345 and do a sepearte measurement for each initalization. Collect the results in a small table and briefly discuss. As always, report compiler, version, and flags. Submit your code to the SVN.

3. *Optimization Blockers (40 pts)* Code needed

   Download, extract and inspect the code. Your task is to optimize the function called superslow (guess why it's called like this?) in the file **comp.c**. The function runs over an $n \times n$ matrix and performs some computation on each element. In its current implementation, *superslow* involves several optimization blockers. Your task is to optimize the code.

   Edit the Makefile if needed (architecture flags specifying your processor). Running `make` and then the generated executable verifies the code and outputs the performance (the flop count is underestimated, since the trigometric functions are ignored) of superslow. Proceed as follows

   (a) Identify optimization blockers discussed in the lecture and remove them.

   (b) For every optimization you perform, create a new function in `comp.c` that has the same signature and register it to the timing framework through the *register_function* procedure in *comp.c* Let it run and, if it verifies, determine the performance.

   (c) In the end, the innermost loop should be free of any procedure calls and operations other than adds and mults.

   (d) When done, rerun all code versions also with optimization flags turned off ($-O0$ in the Makefile).

   (e) Create a table with the performance numbers. Two rows (optimization flags, no optimization flags) and as many columns as versions of superslow. Briefly discuss the table.

   (f) Submit your `comp.c` to the SVN

   What speedup do you achieve?

   Solution: A reasonable transformation which would yield full points can be found here: Example Solution

4. *Locality of Matrix Transposition(10 pts)*

   Consider the following straightforward C code for transposing an $M \times N$ matrix A.

```
double A[M][N], B[N][M];

for (int i = 0; i < M; i++) {
      for (int j = 0; j < N; j++) {
            B[j][i] = A[i][j];
      }
}
```

   Inspecting the data accesses, where do you see

   (a) Temporal locality?

   (b) Spatial locality?

   Solution: Within the data every element is only accessed once and therefor shows no temporal locality. Spatial locality can be seen on the accesses on A which is traversed in a stride-1 fashion. B is traversed with a stride of N, so spatial locality depends on how we define "nearby" and on how big N is.

5. *Locality of a Triangular Solver (10 pts)*

   Consider the following C code for solving a triangular system of equations Lx = b, where L is a non singular, lower triangular matrix of size $N \times N$, while b and x are vectors of size N. The function overwrites b with the solution.

```
double L[N][N], b[N];

for (int i = 0; i < N-1; i++) {
     b[i] = b[i]/L[i][i];
```

```
        for (int j = i+1; j < N; j++) {
            b[j] -= b[i]*L[j][i];
        }
    }
    b[N-1] = b[N-1]/L[N-1][N-1];
```

Inspecting the data accesses, where do you see

(a) Temporal locality?

(b) Spatial locality?

Solution: b[i] is accessed multiple times in every iteration of the inner loop and thus yields temporal locality. b[j] is accessed in a stride-1 fashion, thus exhibits spatial locality L[j][i] is walked in stride N fashion - therefore its again debatable whether this is spatial locality.