263-2300-00: How To Write Fast Numerical Code
Assignment 7: 100 points
Due Date: Tue May 8 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring12/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/s12-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. *Warmup (30 pts)*

   SAXPY is a BLAS 1 function that computes $y = \alpha x + y$ on single precision floats, where x, y are vectors and α a scalar. The file `saxpy.c` contains the scalar code for computing this operation in the function `saxpy`. Your task is to implement the a vectorized version of this procedure, called `vec_saxpy` in the same file, using SSE intrinsics. You can assume all arrays are 16-byte aligned.

   Compile using the provided makefile and run the code.

   Compute the vectorization efficiency.

   Reminder: In class (Lecture 15) we defined vectorization efficiency as the ratio

   $$V_{eff} = \frac{\text{Op count scalar code}}{\text{Op count vectorized code}}, \tag{1}$$

   where the op count of the vectorized code includes all those instructions that perform operations and that are used to manage the positioning of the data in the registers (shuffle, unpack, etc...) with the exception of loads and stores. Note that certain intrinsics create a sequence of two or more instructions.

2. *Medium (30 pts)*

   The formula for multiplying two complex numbers is given as:

   $$(a + ib)(c + id) = (ac - bd) + i(ad + bc), \quad a, b, c, d \in \mathbb{R} \text{ and } i = \sqrt{-1} \tag{2}$$

   Scalar C code for computing the pointwise multiplication of two arrays of complex numbers is provided in `complexmul.c` in the function `cplxmul`. The function assumes that complex numbers are stored in the so-called interleaved form. This means a complex vector of length n is represented as array of length $2n$, in which the entries are alternating real and imaginary parts. For example, the vector $(2 + 4i, 1 + 3i)$ would be represented as $[2, 4, 1, 3]$.

   Your task is to implement *vec_cplxmul*, the vectorized version of the given function. You can assume all arrays are 16-byte aligned.

   Compile, run and compute the vectorization efficiency of your code.

3. *Puzzle (40 pts)*

   Your task is now to vectorize the code for a 5×5 MVM $y = Ax$: Implement the function `vec_mvm5` in `mvm5.c` using SSE intrinsics; do not use any unaligned load or store instruction. You can assume all arrays are 16-byte aligned.

   Compile, run, and determine the vectorization efficiency.

Note: In all the above exercises you need to achieve a high enough vectorization efficiency to obtain full points. Always verify the code using the provided routines.