263-2300-00: How To Write Fast Numerical Code
Assignment 4: 70 points
Due Date: Thu, May 12th, 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring16/course.html
Questions: fastcode@lists.inf.ethz.ch

Submission instructions (read carefully):

- (Submission)
  Homework is submitted through the Moodle system https://moodle-app2.let.ethz.ch/course/view.php?id=2125. Before submission, you must enroll in the Moodle course.
- (Late policy)
  You have 3 late days, but can use at most 2 on one homework, meaning submit latest 48 hours after the due time. For example, submitting 1 hour late costs 1 late day. Note that each homework will be available for submission on the Moodle system 2 days after the deadline. However, if the accumulated time of the previous homework submissions exceeds 3 days, the homework will not count.
- (Formats)
  If you use programs (such as MS-Word or Latex) to create your assignment, convert it to PDF and name it homework.pdf. When submitting more than one file, make sure you create a zip archive that contains all related files, and does not exceed 10 MB. Handwritten parts can be scanned and included or brought (in time) to Alen's or Gagandeep's office. Late homeworks have to be submitted electronically by email to the fastcode mailing list.
- (Plots)
  For plots/benchmarks, **provide (concise) necessary information for the experimental setup (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 from the lecture.
- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. (40 pts) Multiplication of Complex Vectors
   Implement a pointwise multiplication of two complex vectors $(c_0, \ldots, c_{n1})$ of length n by a second complex vector of the same length. Each array of complex numbers is represented as an array of twice the length containing floats. This array contains alternating the real and imaginary parts:

   $$(Re(c_0), Im(c_0), \ldots, Re(c_{n1}), Im(c_{n1}))$$

   This format is called *interleavedcomplex* format. Assume that the code will run on 3 different CPUs:

   - Intel Core2 Duo CPU, with SSE 4.2 support
   - Intel Core i7 CPU, with AVX support
   - Intel Core i7 CPU, with AVX 2.0 support and FMA.

   Code skeleton is available here. Modify the code skeleton such that you provide 3 different versions for point-wise multiplication, using SSE, AVX, and FMA instructions. The skeleton provides validation infrastructure, with 3 test groups and requires binary compatibility between the base SISD version and your SIMD solution (i.e. floats are compared bit-wise for equality). Your tasks:

   (a) Try to validate your results and submit `pointwise_SSE.c`, `pointwise_AVX.c` and `pointwise_FMA.c`.
   (b) Can you validate `pointwise_FMA.c` for 'Test Group 3'? Why is that?
   (c) Report your results, and the obtained speed up.
   (d) Why is 'Test Group 3' slower for both SIMD and SISD versions? How can you avoid that?

Note that the skeleton is written such that it compiles to machines that do not support AVX/FMA. If you do not have a machine that supports AVX or FMA, use the ISG public student labs, or the login server `optimus7.inf.ethz.ch`. If the CPU that you use, has FMA or AVX support, but the feature bits are not automatically detected, you can force the use of SSE, AVX or FMA by adding extra argument on the console:

```
./bin/pointwise 1 # (for SSE)
./bin/pointwise 2 # (for AVX)
./bin/pointwise 3 # (for FMA)
```

Solution:

(a) One possible solution of the pointwise multiplication can be found here.

(b) `pointwise_FMA.c` for 'Test Group 3' can not be validated. This is because 'Test Group 3' deals with special kind of floats, called *denormals* (or subnormals in IEEE 754-2008). The representation of denormals results in an exponent that is below the minimum exponent, and as such they have mantisa which is prefixed with leading zeros. On the other hand, replacing one multiply instruction and one add instruction with a single FMA instruction changes associativity. Since FMA in general receives one rounding instead of two roundings in the use of the combined instructions, FMA yields more accurate results. This accuraccy particularly affects the mantisa, and thus binary compatibility can not be achieved on denormals.

(c) Results on an Intel(R) Xeon(R) CPU E3-1285L v3 @ 3.10GHz are available here.

(d) 'Test Group 3' is slower for both SIMD and SISD because denormals require special treatment in hardware. On modern processors handling an operation on denormals can cost up to 100 cycles. To avoid this, Intel provides special flags in their CPUs, called denormals-are-zero (DAZ) and flush-to-zero (FTZ) flags. Those modes can be enabled by setting the control register using `_mm_setcsr` instrinsics. The obtained results above, enable those flags in the initialization segment of the code.

2. (25 pts) MVM
Implement a vectorized version of an n x n matrix vector multiplication. Use AVX only and assume that the matrix and vector contain real numbers, represented as single precision floats. Modify the skeleton available here and submit only `mvm.c`. This exercise is only about vectorization (i.e., not about blocking for cache etc.) which should be reasonably efficient.

Solution:
The Solution code for MVM can be found here.