263-0007-00: Advanced Systems Lab
Assignment 3: 100 points
Due Date: Th, March 30st, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2023/
Questions: fastcode@lists.inf.ethz.ch

Academic integrity:
All homeworks in this course are single-student homeworks. The work must be all your own. Do not copy any parts of any of the homeworks from anyone including the web. Do not look at other students' code, papers, or exams. Do not make any parts of your homework available to anyone, and make sure no one can read your files. The university policies on academic integrity will be applied rigorously.

Submission instructions (read carefully):

- (Submission)
  Homework is submitted through the Moodle system and through Code Expert for coding exercises. The enrollment link for Code Expert is https://expert.ethz.ch/enroll/SS23/asl.
- (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 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.
- (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.
- (Code)
  The code has to be submitted through Code Expert https://expert.ethz.ch/mycourses/SS23/asl.

Instructions

In this homework, you will have to implement some computations with vector intrinsics using the **AVX** family of intrinsics (AVX, AVX2 and FMA). It is **not allowed** to use SSE intrinsics. Your implementations have to be completely vectorized. Thus, you should use vector intrinsics for reading from memory, performing the necessary computations, and for writing the result back to memory. **Implementations without vectorization will not be counted as valid**. We will check this separately outside Code Expert. The only exception where it is allowed not to use intrinsics is in small computations outside loop boundaries (e.g. to handle the remaining elements after loop unrolling).

The following information applies to all exercises:

- You may apply any optimization that produces the same result in exact arithmetic.
- Similar to the previous homework, the code provided in Code Expert allows you to register functions which will be timed in a microbenchmark fashion.
- You can create a new function and register it to the timing framework through the *register_function* function. Let it run and, if it verifies, it will print the measured runtime in cycle.
- Implement in function *maxperformance* the implementation that achieves the best runtime. This is the one that will be autograded by Code Expert.

- Code Expert compiles the code using GCC 11.2.1 with flags `-O3 -march=skylake -fno-tree-vectorize -ffp-contract=off`. It is **not allowed** to use pragmas to modify the compilation environment.
- The CPU running the jobs submitted to Code Expert is an Intel Xeon Silver 4210 Processor.
- Don't forget to click on the "Submit" button when you finish an exercise. There is no need to submit anything in Moodle for this homework.

Exercises

1. Quaternions (25 pts)

   In this exercise, we consider the following function that computes the outer product $A = x \otimes y$ of two vectors x, y with quaternion numbers.

```
#include "quaternion.h"
void slow_performance1(quaternion_t x[N], quaternion_t y[N], quaternion_t A[N][N]) {
    for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
            A[i][j] = mul(x[i], y[j]);
}
```

   A quaternion number is a generalization of a complex number with three imaginary parts instead of only one. More precisely, quaternions are represented in the form $a + bi + cj + dk$ where a, b, c, d are real numbers and the following properties hold:

$$i^2 = j^2 = k^2 = ijk = -1, \quad ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j.$$

   We provide the header file `quaternion.h` implementing the addition and multiplication operations on quaternion numbers. Your task is to optimize this computation using the AVX family of intrinsics. Note that using the structure `quaternion_t`, the quaternion numbers in vectors x, y are stored in *interleaved* format: $[a_0, b_0, c_0, d_0, a_1, b_1, c_1, d_1, ...]$, where each quadruplet (a_n, b_n, c_n, d_n) represents the quaternion number $a_n + b_n i + c_n j + d_n k$. For this exercise, you can assume that $N = 2$.

2. *Complex vector (35 pts)*

   In this exercise, we consider the following computation that takes as input an array x of complex numbers and an array y of real numbers (represented using double precision).

```
// Precondition: 0 <= y[i] < 1.0
void slow_performance1(complex_t *x, double *y, int n) {
    for (int i = 0; i < n; i++) {
        unsigned int k = floor(4.0*y[i]);
        switch (k) {
            case 0:  y[i] += fmin(re(sqr(x[i])), im(sqr(x[i]))); break;
            case 1:  y[i] += fmax(re(sqr(x[i])), im(sqr(x[i]))); break;
            default: y[i] += pow(abs(x[i]), 2.0); break;
        }
    }
}
```

   Your task is to optimize this computation using the AVX family of vector intrinsics. Using the structure `complex_t`, the complex numbers in x are stored in *interleaved* format alternating the real and the imaginary part. You can assume that $0 \leq y_i < 1$ and $n = 1024$.

3. *FIR Filter (40 pts)*

   In this exercise we consider a finite-impulse-response filter (FIR) defined as:

$$y_i = \sum_{k=0}^{m-1} (i + k + 1) \cdot h_k \cdot |x_{i+(m-1)-k}|.$$

   Your task is to optimize this computation using the AVX family of vector intrinsics. Assume that $m = 4$.