263-0007-00: Advanced Systems Lab

Assignment 3: 100 points
Due Date: Th, April 1st, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2021/
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 https://moodle-app2.let.ethz.ch/course/view.php?id=14942 and through Code Expert https://expert.ethz.ch/mycourses/SS21/asl for coding exercises.
- (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/SS21/asl.
- (Neatness)
  5% of the points in a homework are given for neatness.

Instructions

In this homework, you will have to implement some computations with vector intrinsics using the **AVX2** (with FMA) instruction set. 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.

- The Code Expert system compiles the code using GCC 8.3.1 and flags `-O3 -fno-tree-vectorize -march=skylake`. It is **not allowed** to use pragmas to modify the compilation environment.
- 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.
- The CPU running the jobs submitted to Code Expert is an Intel Xeon Silver 4210 Processor.

Exercises

1. *Warm up: Complex conversion (15 pts)*

   In this exercise, we consider the following computation that reads an array x of complex numbers stored in interleaved format and stores the real and the imaginary part into different arrays:

```
void complex_conversion(double *x, double* re, double* im, int N) {
  int k = 0;
  for (int i = 0; i < N; i+=2) {
      re[k] = x[i];
      im[k] = x[i+1];
      k++;
  }
}
```

   Your task is to implement this computation with vector intrinsics using the AVX2 (with FMA) instruction set. You can assume that N is divisible by two.

2. *Vector addition (35 pts)*

   In this exercise, we consider the following computation:

```
 1  struct vector_t {
 2    double*   val;
 3    uint64_t*  id;
 4  };
 5
 6  void vector_addition(vector_t x, vector_t y, vector_t z, int n) {
 7    for (int i = 0; i < n; i++) {
 8      if (x.id[i] == y.id[i]) {
 9        z.val[i] = x.val[i] + y.val[i];
10        z.id[i]  = x.id[i];
11      }
12      else if (fabs(x.val[i]) > fabs(y.val[i])) {
13        z.val[i]  = x.val[i];
14        z.id[i]   = x.id[i];
15        z.val[n] += fabs(y.val[i]);
16      }
17      else {
18        z.val[i]  = y.val[i];
19        z.id[i]   = y.id[i];
20        z.val[n] += fabs(x.val[i]);
21      }
22    }
23  }
```

   Here, each vector consists of an array of doubles where each element has an identifier. Given two vectors x and y, the computation adds their i-th elements only if they have the same identifier (lines 8–10). Otherwise, only the element with the largest magnitude is selected and the smallest element is added to the n-th position of vector z (lines 12–20). Your task is to implement this computation with vector intrinsics using the AVX2 (with FMA) instruction set. Optimize it as much as you can.

3. *FIR Filter (50 pts)*

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

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

   Your task is to implement this computation with vector intrinsics using the AVX2 (with FMA) instruction set. Optimize it as much as you can. You can assume that $m = 4$.