263-0007-00: Advanced Systems Lab
Assignment 2: 80 points
Due Date: Th, March 18th, 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. The enrollment link is https://expert.ethz.ch/enroll/SS21/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 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.
- (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)
  When compiling the final code, ensure that you use optimization flags (e.g. for GCC use the flag "-O3") unless indicated otherwise.
- (Neatness)
  5 points in this homework are given for neatness.

Exercises:

1. *Short project info (5 pts)*
   Go to the list of milestones for the projects. If you have not done that yet, please register your project there. Read through the different points and fill in the first two with the following about your project (be brief):

   Point 1) An exact (as much as possible) but also short, problem specification.

   For example for MMM, 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 $C = AB \in \mathbb{R}^{n \times m}$.

   Give the name of the algorithm you plan to consider for the problem and a precise reference (e.g., a link to a publication plus the page number) that explains it.

   Point 2) A very short explanation of what kind of code already exists and in which language it is written.

2. *Optimization Blockers (30 pts)*

   In this exercise, we consider the following short computation that is available in Code Expert:

```
#define C1 0.2
#define C2 0.3
void slowperformance1(float *x, float *y, float *z, int n) {
    for (int i = 0; i < n - 2; i++) {
        x[i]     = x[i] / M_SQRT2 + y[i] * C1;
        x[i+1]  += z[(i % 4) * 10] * C2;
        x[i+2]  += sin((2 * M_PI * i) / 3) * y[i+2];
    }
}
```

   This is part of the supplied code in Code Expert:

   - Read and understand the code. It enables you to register functions with the same signature, which will be timed in a microbenchmark fashion.
   - Create new functions where you perform optimizations to improve the runtime. For example, loop unrolling and scalar replacement as discussed in the lecture, strength reduction, inlining, and others.
   - You may apply any optimization that produces the same result in exact arithmetic.
   - For every optimization you perform, create a new function in `comp.cpp` that has the same signature as *slowperformance1* and register it to the timing framework through the *register_function* function in `comp.cpp`. Let it run and, if it verifies, it will print the runtime in cycles.
   - Implement in function *maxperformance* the implementation that achieves the best runtime. This is the one that will be autograded by Code Expert.
   - For this task, the Code Expert system compiles the code using GCC 8.3.1 with the following flags: `-O3 -fno-tree-vectorize`. Note that with these flags vectorization and FMAs are disabled.
   - It is not allowed to use vector instrinsics or FMA to speedup your implementation.
   - Hint: The code above may present additional overhead due to casts between double precision and single precision floating point. It is advisable to avoid these casts by using only one precision accross all operations.

   Discussion:

   (a) Create a table with the runtime numbers of each new function that you created (include at least 2). Briefly discuss the table explaining the optimizations applied in each step.

   (b) What is the speedup of function *maxperformance* compared to *slowperformance1*?

   (c) What is the performance in flops/cycle of your function *maxperformance*.

3. *Microbenchmarks(40 pts)*

   Your task is to write a program (without vector instructions, i.e., standard C) in Code Expert that benchmarks the latency and inverse throughput (also called "gap") of floating point addition and division on doubles. In addition, the latency and gap of function $f(x) = \frac{x}{1+x^2}$. We provide the implementation of $f(x)$ in `foo.h`. More specifically:

   - Read and understand the code given in Code Expert.
   - Implement the functions provided in the skeleton in file `microbenchmark.cpp`:

```
void    initialize_microbenchmark_data (microbenchmark_mode_t mode);
double  microbenchmark_get_add_latency();
double  microbenchmark_get_add_gap();
double  microbenchmark_get_div_latency();
double  microbenchmark_get_div_gap();
double  microbenchmark_get_foo_latency();
double  microbenchmark_get_foo_gap();
```

- You can use the `initialize_microbenchmark_data` function for any kind of initialization that you may need (e.g. for initializing the input values).
- Note that the latency and gap of floating point division can vary depending on its inputs. Thus, you are also required to find the minimum latency and gap for division and function $f(x)$. Hint: You can try using values where performing a division becomes trivial.
- It is not allowed to manually inline the function in `foo.h` into the implementation of your microbenchmarks.

Additional information:

- Our Code Expert system already has Turbo Boost disabled. However, note that CPUs may throttle their frequency below the nominal frequency. To ensure that the CPU is not throttled down when running the experiments, one can **warm up** the CPU before timing them.
- For this task, our Code Expert system uses GCC 8.3.1 to compile the code with the following flags: `-O3 -fno-tree-vectorize -march=skylake`. Note that with these flags vectorization is disabled but FMAs are enabled.

Discussion:

(a) Do the latency and gap of floating point addition and division match what is in the Intel Optimization Manual? If no, explain why. (You can also check Agner's Table).

(b) Based on the dependency, latency and gap information of the floating point operations, is the measured latency and gap of function $f(x)$ close to what you would expect? Justify your answer.

(c) Will the latency and gap of $f(x)$ change if we compile the code with flags `-O3 -fno-tree-vectorize` (i.e., with FMAs disabled)? Justify your answer and state the expected latency and gap in case you think it will change.

Information regarding Code Expert

- Don't forget to click on the "Submit" button when you finish an exercise.
- The CPU running the programs submitted to Code Expert is an Intel Xeon Silver 4210 Processor. This is a Cascade Lake processor but you can assume the same latency and throughput information as Skylake.