263-0007-00: Advanced Systems Lab
Assignment 2: 80 points
Due Date: Th, March 12th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2020/
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=10968 and through Code Expert https://expert.ethz.ch/mycourses/SS20/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/SS20/asl.
- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. *Short project info (10 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 (25 pts)*

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

```
void slowperformance1(double *w, double *x, double *y, double *z, int n) {
  for(int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
          if ((i + j) % 2) {
              z[i] += 1.0 / (x[i*n + j] * sqrt(w[i]));
          }
          else {
              z[i] = compute(w[i], y[i*n + j], z[i]);
          }
          z[i] *= x[i*n + j];
      }
  }
}
```

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.
- Count a sqrt operation as a single floating point operation.
- 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.
- The array w is guaranteed to contain only positive values.
- 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 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.

Discussion:

(a) Create a table with the runtime numbers of each function that you created. Briefly discuss the table.

(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(45 pts)*

In Code Expert, we provide three functions in file `sigmoid.h` that implement a floating point square root instruction (*sqrtsd*) and two commonly used activation functions in Neural Networks (*sigmoid1* and *sigmoid2*). Your task is to write a program (without vector instructions, i.e., standard C) in Code Expert that benchmarks the maximum and minimum latency and inverse throughput (also called "gap") of *sqrtsd* and *sigmoid1*. In addition, the latency and gap of *sigmoid2* for inputs 1.0 and 0.0. More specifically:

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

```
void    initialize_microbenchmark_data (microbenchmark_mode_t mode);
double  microbenchmark_get_sqrt_latency();
double  microbenchmark_get_sqrt_gap();
double  microbenchmark_get_sigmoid1_latency();
double  microbenchmark_get_sigmoid1_gap();
double  microbenchmark_get_sigmoid2_latency();
double  microbenchmark_get_sigmoid2_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 for the sigmoid functions).
- Function *microbenchmark_get_sqrt_latency* should return the measured latency of the *sqrtsd* function. Analogously, the other functions should return the latency (or gap) of the function implied by its name. The gap is the inverse of the throughput. The latency and gap have to be measured in cycles.
- Note that the latency and gap of some instructions (e.g. square root) can vary depending on their input. Thus, your task is to find the minimum and maximum latency and gap of *sqrtsd* and *sigmoid1*. In addition, the latency and gap of *sigmoid2* for inputs 1.0 and 0.0.
- It is not allowed to manually inline the functions in `sigmoid.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 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.
- The CPU running the programs submitted in Code Expert is an Intel Xeon Silver 4210 Processor. This is a Cascade Lake processor but the latency and throughput of all instructions are the same as Skylake.

Discussion:

(a) Do the latency and gap of the square root instruction match what is in the Intel Optimization Manual?

(b) Based on the dependency, latency and gap information of the operations used to implement function *sigmoid1*. Is the measured latency and gap of the *sigmoid1* close to what you would expect?