263-0007-00: Advanced Systems Lab
Assignment 2: 80 points
Due Date: Th, March 12th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2026/
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/SS26/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. Note that in this homework we also consider the code submitted in CodeExpert 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 together with your team. It is enough if only one member of the team submits this in the project system. Consider the following about your project while filling the points (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 part of the supplied code in Code Expert:

```
void slow_performance1(Particle* p, int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (i == j) {
        continue;
      }
      double dist = compute_distance(p[i], p[j]);
      double dist_sq = dist * dist;
      if (dist > 0.01) {
        double interaction_energy = compute_interaction(p[i], p[j]);
        double force_mag;
        if ((i+j)%2 == 0) {
          force_mag = (p[i].mass * p[j].mass) / (dist_sq * dist_sq);
        } else {
          force_mag = -(p[i].mass * p[j].mass) / dist;
        }
        force_mag = -force_mag;

        p[i].energy += interaction_energy;
        p[i].force += force_mag;
      }
    }
  }
}
```

Do the following:

- 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, strength reduction, inlining, removing function calls, precomputations, 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 *slow_performance1* 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.
- **Important:** the environment on Code Expert has limitations on execution time and memory usage. You may need to run some implementations individually depending on how optimized they are. You can do this by simply commenting the slow implementations out in `register_functions`.
- For this task, the Code Expert system compiles the code using GCC 11.2.1 with the following flags: `-O3 -march=skylake -mno-fma -fno-tree-vectorize`. Note that with these flags vectorization and FMAs are disabled. It is also not allowed to use pragmas that modify the compilation environment.
- It is not allowed to use vector intrinsics or FMAs to speedup your implementation.
- In case you need it, you can consider the `std::log` function to be around 30 flops. See the FAQs.
- The implementations of the `Particle` struct, and the functions that operate on it, are given in `particles.cpp` and `particles.h`. You can make the following assumptions:
  (a) There are 100 particles,
  (b) The x and y coordinates of the particles are integers in $[-10, 10]$. In other words, they are distributed in a 21×21 grid (441 total positions).
  (c) No two particles are in the same spot in the grid.

Discussion:

(a) Create a table with the runtime numbers of each new function that you created (include at least 3). Briefly discuss the table explaining the optimizations applied in each step. Mention also the maximum speedup that you achieved.

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

(c) Consider the theoretical peak performance for one core, without SIMD vector instructions and without FMAs of the machine running the programs submitted to Code Expert. What percentage of this theoretical peak performance did you achieve?

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 reciprocal throughput of the fused multiply-add and square root operations on doubles. In addition, the latency and reciprocal throughput of the function $f(a, b) = \sqrt{a^3 + b}$. We provide the implementation of $f(a, b)$ 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_fma_latency();
double  microbenchmark_get_fma_rec_tp();
double  microbenchmark_get_sqrt_latency();
double  microbenchmark_get_sqrt_rec_tp();
double  microbenchmark_get_foo_latency();
double  microbenchmark_get_foo_rec_tp();
```

- 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 reciprocal throughput of floating-point square root can vary depending on their inputs. Thus, you are also required to find the minimum latency and reciprocal throughput for square root and function $f(a, b)$. Hint: You can try using values where performing those operations becomes trivial.
- It is not allowed to manually inline the function in `foo.h` into the implementation of your microbenchmarks.
- Make sure that your benchmarks yield stable measurements between runs, i.e the measurements should not oscillate between correct and incorrect. After the deadline, we will run each submission ten times and evaluate the score based on the run with the fewest correct measurements. For instance, if your submission initially achieves a 10/10 score, but upon rerunning it we observe a 6/10, the final score will be recorded as 6/10.

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. This can be done, for example, by running a loop a few thousand times. Find a way to avoid the compiler optimizing the code away.

```
int warmup_iter = ...; // We use a value in the thousands.
for (int i = 0; i < warmup_iter; ++i) {
  bar(); // some user defined operation
  // Be careful, if the compiler notices that this
  // loop does nothing, then it will remove this piece of code.
}
```

- For this task, our Code Expert system uses GCC 11.2.1 to compile the code with the following flags: `-O3 -fno-tree-vectorize -march=skylake -mfma -fno-math-errno`. The `-fno-math-errno` flag is used to guarantee that the `sqrt` function call is converted to its respective instruction.

Discussion:

(a) Do the latency and reciprocal throughput of floating point fma and square root match what is in the Intel Optimization Manual? If no, explain why. (You can also check Agner's Table, or uops).

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

(c) Assume that we compile with FMA disabled, i.e using the `-mno-fma` flag instead of `-mfma`. Will the latency and reciprocal throughput of $f(a)$ change? Justify your answer and state the expected latency and reciprocal throughput in case you think it will change.

(d) Assume that we implement two variations of $f(a, b)$ as follows:

$$f_2(a,b) = a\sqrt{\sqrt{a}+b}, \quad \text{and} \quad f_3(a,b) = \sqrt{1/(a+b)}.$$

Will the latency and reciprocal throughput of $f_2(a, b)$ and $f_3(a, b)$ change compared to the latency and reciprocal throughput of $f(a, b)$? Justify your answer and state the expected latency and reciprocal throughput in case you think it will change. Consider the `-mfma` case.

Information regarding Code Expert

- Don't forget to click on the **Submit** button when you finish an exercise.
- Click on the **Test** button (depicted with a flask symbol) next to the terminal in Code Expert to run and test your code before submitting. The test button has a slightly different functionality than the run button and allows you to check your score before submitting.
- 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.