263-2300-00: How To Write Fast Numerical Code
Assignment 1: 100 points
Due Date: Th, March 7th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2019/
Questions: fastcode@lists.inf.ethz.ch

Submission instructions (read carefully):

- (Submission)
  Homework is submitted through the Moodle system https://moodle-app2.let.ethz.ch/course/view.php?id=10968. Before submission, you must enroll in the Moodle course.
- (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 or brought (in time) to Alen's, Tyler's or Gagandeep's office.
- (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").
- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. (20 pts) Get to know your machine
   Determine and create a table for the following microarchitectural parameters of your computer:

   (a) Processor manufacturer, name, and number.

   (b) Number of CPU logical and physical cores.

   (c) CPU-core frequency.

   (d) CPU maximum frequency. Does your CPU support Turbo Boost or a similar technology?

   For one core and **without** using SIMD vector instructions determine:

   (d) Latency [cycles] and throughput [ops/cycle] for floating point additions.

   (e) Latency [cycles] and throughput [ops/cycle] for floating point multiplications.

   (f) Latency [cycles] and throughput [ops/cycle] for floating point divisions.

   (g) Latency [cycles] and throughput [ops/cycle] for fused multiply-add (FMA) operations (if supported).

   (h) Latency [cycles] and throughput [ops/cycle] for converting a double-precision floating-point element to a signed 32-bit integer.

   (i) Maximum theoretical floating point peak performance in both flop/cycle and Gflop/s.

Notes:

- Intel calls throughput what is in reality the gap = 1/throughput.
- The manufacturer's website will contain information about the on-chip details. (e.g. Intel 64 and IA-32 Architectures Optimization Reference Manual).
- On Unix/Linux systems, typing `cat /proc/cpuinfo` in a shell will give you enough information about your CPU for you to be able to find an appropriate manual for it on the manufacturer's website (typically AMD or Intel).
- For Windows 7/10 "Control Panel/System and Security/System" will show you your CPU, for more info "CPU-Z" will give a very detailed report on the machine configuration.
- For Mac OS X there is "MacCPUID".
- Throughout this course, we will consider the FMA instruction as two floating point operations.

2. (10 pts) Cost analysis Consider the following algorithm that solves the linear system $Lx = b$ for x given L and b, where L is an $n \times n$ lower-triangular matrix:

```
1 void lower_triangular_solve (double L[], double x[], double b[], int N)  {
2     for(int i = 0; i < N; i++) {
3         double el10_dot_x0 = 0.0;
4         for(int j = 0; j < i; j++) {
5             el10_dot_x0 += L[i + j*N] * x[j];
6         }
7         x[i] = (b[i] - el10_dot_x0) / L[i + i*N];
8     }
9 }
```

   (a) Define a suitable detailed floating point cost measure $C(n)$.

   (b) Compute the cost $C(n)$ of the function `lower_triangular_solve`.

3. (25 pts) Matrix-vector multiplication

   In this exercise, we provide a C source file for multplying an $n \times n$ matrix with a vector and a C header file. The matrix and vectors are represented in double precision. to time the matrix-vector multiplication using different methods under Windows and Linux (for x86 compatible systems).

   (a) Inspect and understand the code.

   (b) Determine the exact number of (floating point) additions and multiplications performed by the compute() function in `mvm.c`.

   (c) Using your computer, compile and run the code. Compile with the highest level of optimization provided by your compiler (with GCC, compile with the flag "-O3"). A modern compiler will automatically vectorize this very simple routine. Ensure you get consistent timings between timers and for at least two consecutive executions.

   (d) Then, for all square matrices of sizes n between 200 and 4000, in increments of 200, create a plot for the following quantities (one plot per quantity, so 3 plots total). n is on the x-axis and on the y-axis is, respectively,

      i. Runtime (in cycles).

      ii. Performance (in flops/cycle).

      iii. Using the data from exercise 1, percentage of the peak performance reached.

   (e) Briefly discuss your plots.

4. (20 pts) Performance Analysis

Assume that the elements of vectors x, y, u and z of length n are combined as follows:

$$z_i = z_i + u_i \cdot u_i + x_i \cdot y_i \cdot z_i$$

(a) Write a C/C++ compute() function that performs the computation described above on arrays of doubles. Save the file as combine.c(pp).

(b) Within the same file create a benchmarking infrastructure based on the timing function that produces the most consistent results in Exercise 3.

(c) Then, for all two-power sizes $n = 2^4, \ldots, 2^{23}$ create performance plot with n on the x-axis (choose logarithmic scale) and performance (in flops/cycle) on the y-axis. Create two series such that the first has all optimization flags disabled, and the second series has the major optimizations flags enabled (including vectorization). Randomly initialize all arrays. For all n repeat your measurements 30 times reporting the median in your plot.

(d) If you have an Intel processor, run the same tests again, but make sure that Intel Turbo Boost is disabled (or enabled if the previous plot was generated with Turbo Boost disabled).

(e) Briefly explain eventual performance variations in your plot and the effects of Turbo Boost.

5. (20 pts) Bounds

Consider the three artificial computations below. The functions operate on input arrays of length N and store the results in an output array of length N:

```
1  void artcomp1(float alpha, float x[], float y[], float z[], int N) {
2    for (int i = 0; i < N; i++)
3      y[i] = alpha * (x[i] + y[i] + z[i] + 1.0);
4  }
5  void artcomp2(float alpha, float x[], float y[], int N) {
6    for (int i = 0; i < N; i++) {
7      y[i] = alpha / x[i];
8    }
9  }
10 void artcomp3(float x[], float y[], float z[], int N) {
11   //Assume 3 divides N
12   for (int i = 0; i < N; i += 3) {
13     y[i] = x[i] * z[i];
14     y[i + 1] += x[i + 1] * z[i + 1];
15     y[i + 2] = x[i + 2] + z[i + 2];
16   }
17 }
```

We consider a Core i7 CPU based on a Haswell processor. As seen in the lecture, it offers FMA instructions (as part of AVX2) that compute `y = a * x + b` on floating point numbers. Consider the information from the lecture slides on the throughput of the according operations. Assume that divisions are performed with the regular div on Port 0. Assume the bandwidths that are given in the additional material from the lecture: Abstracted Microarchitecture

(a) Determine the exact cost (in flops) of each function.

(b) Determine an upper bound on the operational intensity of each function. Assume empty caches and consider only reads but note: arrays that are only written are also read and the read should be included.

(c) Consider only one core and determine, for each function, a hard lower bound (not asymptotic) on the runtime (measured in cycles), based on:

i. The op count. Assume that the code is compiled using `gcc` with the following flags: `-fno-tree-vectorize -mfma -march=core-avx2 -O3` and that FMAs are used as much as possible. Be aware that the lower bound is also affected by the available ports offered for the computation (see lecture slides).

ii. Loads, for each of the following cases: All floating point data is L1-resident, L2-resident, and RAM-resident. Consider best case scenario (peak bandwidth).

How to disable Intel Turbo Boost

Intel Turbo Boost is a technology implemented by Intel in certain versions of its processors that enables the processor to run above its base operating frequency via dynamic control of the processor's clock rate. It is activated when the operating system requests the highest performance state of the processor.

BIOS

Intel Turbo Boost Technology is typically enabled by default. You can only disable and enable the technology through a switch in the BIOS. No other user controllable settings are available. Once enabled, Intel Turbo Boost Technology works automatically under operating system control. When access to BIOS is not available, few workarounds are possible:

Linux

Linux does not provide interface to disable Turbo Boost. One alternative, that works, is disabling Turbo Boost by writing into MSR registers. Assuming 2 cores, the following should work:

```
wrmsr -p0 0x1a0 0x4000850089
wrmsr -p1 0x1a0 0x4000850089
```

To enable it:

```
wrmsr -p0 0x1a0 0x850089
wrmsr -p1 0x1a0 0x850089
```

This method has been criticized here and, here stating that the OS can circumvent the MSR value, using opportunistic strategy. But so far in our tests, we have observed that Linux conforms to the MSR value. An alternative method would be to use `cpupower`, as explained in the ArchLinux Wiki, as well as the the intel_pstate driver. Unfortunately, we can not confirm deterministic behavior across different kernel versions with this method.

Mac OS X

Disabling Turbo Boost in OS X can be done easily with the Turbo Boost Switcher for OS X. Note that the change is not persistent after restart. The method also writes to the MSR register, and shares the same weaknesses as the Linux approach.

Windows

Windows does not provide any functionality to disable Intel Turbo Boost. The only effective way of disabling is using the BIOS. On some Intel machines however, it is possible to fix the CPU multiplier such that the resulting frequency corresponds to the nominal frequency of the CPU. ThrottleStop provides this functionality with a convenient GUI. "Disable Turbo" will effectively fix the frequency such that it corresponds to a behaviour of a CPU with disabled Turbo Boost.