263-2300-00: How To Write Fast Numerical Code

Assignment 1: 100 points
Due Date: Thu March 8 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring12/course.html
Questions: fastcode@lists.inf.ethz.ch

Submission instructions (read carefully):

- (Submission)
  We set up a SVN Directory for everybody in the course. The Url of your SVN Directory is https://svn.inf.ethz.ch/svn/pueschel/students/trunk/s12-fastcode/YOUR.NETZH.LOGIN/ You should see subdirectory for each homework.
- (Late policy)
  You have 3 late days, but can use at most 2 on one homework. Late submissions have to be emailed to fastcode@lists.inf.ethz.ch.
- (Formats)
  If you use programs (such as MS-Word or Latex) to create your assignment, convert them to PDF and submit to svn in the top level of the respective homework directory. Call it homework.pdf.
- (Plots)
  For plots/benchmarks, be concise, but provide necessary information (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 (lecture 5).
- (Code)
  When compiling the final code, ensure that you use optimization flags. Disable SSE for this exercise when compiling. Under Visual Studio you will find it under Config / Code Generator / Enable Enhanced Instructions (should be off). With gcc their are several flags: use -mno-abm (check the flag), -fno-tree-vectorize should also do the job. Submit all the code you write into the according folders in your SVN directory.
- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. (25pts) Cost analysis

   Consider the following Matlab function. Input and output are both vectors of length n, $n = 2^k$ is assumed to be a two-power.

```
function y = func(x)

n = length(x);
y = zeros(n,1); % allocate the result y which is a column vector

% base case
if n == 1
    y(1) = x(1); return
end

m = n/2;

% recurse
t1 = 2 * func(x(1:m));
t2 = func(x(m+1:n));

for i = 1:m
  y(2*i-1) = t1(i) + t2(i);
  y(2*i) = t1(i) - t2(i);
end

y(1) = y(1) + 1;
return
```

We assume the floating point cost measure $C(n) = (A(n), M(n))$, where $A(n)$ is the number of additions and $M(n)$ the number of multiplications. Compute $C(n)$. Show your work.

Note: Only consider floating point ops.

2. (10pts) 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 cores

   (c) CPU-core frequency

   For one core and without considering SSE/AVX:

   (d) Cycles/issue for floating point additions

   (e) Cycles/issue for floating point multiplications

   (f) Maximum theoretical floating point peak performance (in Gflop/s)

   Tips: 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). The manufacturer's website will contain information about the on-chip details. (e.g. Intel). For Windows 7 "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.

3. (30pts) MMM
   The standard matrix multiplication kernel performs the following operation : $C = AB + C$, where A, B, C are matrices of compatible size. We provide a C source file and a C header file that times this kernel using different methods under Windows and Linux (for x86 compatibles).

   (a) Inspect and understand the code.

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

   (c) Using your computer, compile and run the code (Remember to turn off vectorization!) . Ensure you get consistent timings by at least 2 different timers. You need to setup the timer for your machine by setting the number of runs NUM_RUNS and the machine frequency FREQUENCY. For the number of runs per n use the formula $2 * (ceil(1000/n))^3$.

   (d) Then, for all square matrices of sizes n between 100 and 1500, in increments of 100, 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 seconds)

      ii. performance (in Mflop/s or Gflop/s).

      iii. Using the data from exercise 2, percentage of the peak performance reached.

   (e) Submit your modified code to the SVN and call it also mmm.c

4. (30pts) Bounds
   We consider matrix-vector multiplication of the form $y = A * x + y$, where A is an $n \times n$ square matrix and y and x are vectors of length n. A straightforward implementation uses this double loop:

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
     y[i] = y[i] + A[i][j]*x[j];
```

   On a Core 2, consider only one core and detemine hard lower bounds on the runtime (measured in cycles) based on

(a) The op count (floating point ops only, no vectorization)

(b) Loads, for each of the cases L1-resident, L2-resident, RAM-resident (again only floating point data)

Finally, assume the Core 2 runs at 3 GHz, and

(c) Compute for each of the four lower bounds on the runtime an upper bound for the performance (in Gflop/s).