263-2300-00: How To Write Fast Numerical Code
Assignment 5: 100 points
Due Date: Thu April 5 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring12/course.html
Questions: fastcode@lists.inf.ethz.ch

Exercises:

1. *(Mini-MMM 65 pts)* Code needed
   The goal of this exercise is to implement a high performance mini-MMM (similar to how ATLAS optimizes it) to multiply two square $N_B \times N_B$ matrices (N_B is a parameter), which is then used within MMM in problem 2.

   (a) (By definition) Implement the code that implements MMM directly based on its definition (triple loop implementation), using the ijk loop order. Call this **code0**. We view this code as implementing a mini-MMM.

   (b) (Register blocking) Block into micro-MMMs with $M_U = N_U = 2,\ K_U = 1$. The inner triple loop must have the kij order, as explained in class. Manually unroll the innermost i- and j-loop and perform scalar replacement on this unrolled code and write SSA code. Call this **code1**.

   (c) (Unrolling) Unroll the innermost k-loop by a factor of 2 ($K_U = 2$, which doubles the loop body) and again do scalar replacement (SSA code). Note that the $M_U \times N_U$ block of the resulting matrix is loaded only once outside the innermost k-loop (as explained in class). Assume that 4 divides N_B. This part gives you **code2**.

   (d) (Alternative micro-MMM, following the x86 extended model from class) Now block **code0** for mini-MMM into micro-MMMs with $M_U = 1,\ N_U = 8,\ K_U = 2$. Again, unroll the innermost k,i,j loops and do scalar replacement with SSA. This part gives you **code3**.

   (e) (Best block size N_B) Determine the L1 data cache size C_1 in doubles and its block size B_1, also in doubles. Use the model (inequality) from class (section 2e in the MMM optimization notes) to determine the best (largest) block size N_B for each: **code1**, **code2**, **code3**. Run these three for this block size and report the performance obtained (three numbers) in Gflop/s. Which one is best?

   (f) (Blocking for L2 cache) Now go through the same steps as in the previous part, but this time considering your L2 cache. Measure and report the three performance numbers.

   Your best mini-MMM is the code plus block size that achieved the highest performance among the six in parts 1e and 1f

2. *MMM (15 pts)* Implement an MMM for multiplying two square $n \times n$ matrices assuming N_B divides n, blocked into $N_B \times N_B$ blocks using your best mini-MMM code from exercise 1. This is your **finalcode**. Create a performance plot comparing this code and **code0** (by definition) above for sizes roughly in the range $n = 100, \ldots, 1500$ in steps of roughly 100 (the exact numbers will depend on the N_B you found since you want multiples of N_B). The x-axis shows n; the y-axis performance in Mflop/s or Gflop/s. Briefly discuss the plot.

3. *Roofline (15 pts)* Using the microarchitecture parameters from lecture 3, draw a roofline plot for double precision floating point operations on a Core i7 with AVX. The units for x-axis and y-axis are flops/byte and flops/cycle, respectively. Specifically, the plot should contain 4 lines:

   (a) Upper bound based on peak performance with AVX

   (b) Upper bound based on scalar peak performance

   (c) Upper bound based on the maximal memory bandwidth

   (d) Upper bound based on the maximal bandwidth achievable without spatial locality (e.g., random access of doubles in a very large array)

Provide enough detail (labels etc.) so we can check correctness.

Finally answer the following: What is the minimal operational intensity that a non-vectorizable computation without spatial locality needs to have to be compute bound?

Solution: From the microarchitecture parameters given in lecture 3:

(a) 8 flops/cycle

(b) 2 flops/cycle

(c) Maximal memory bandwidth = 1 double/cycle = 8 bytes/cycle

(d) Maximal bandwidth achievable without spatial locality = $\frac{1}{8}$ double/cycle = 1 byte/cycle

Note that the bandwiths always refer to off-chip data transfers. The roofline plot is provided in Fig. 1.

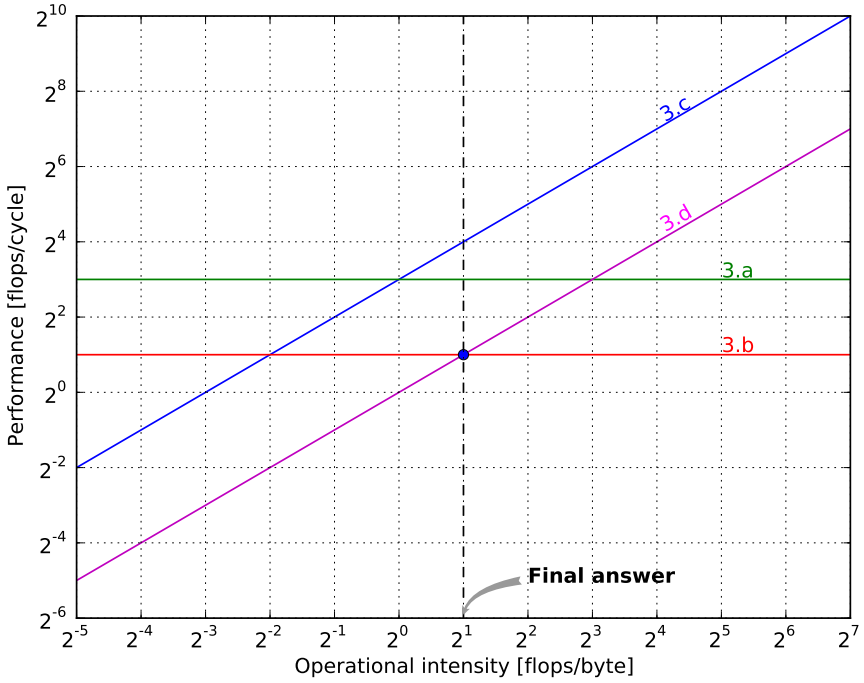

Figure 1: Upper bound roofline plot for double precision floating point operations on the Core i7 with AVX.