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

Assignment 3

Due Date: Thu March 31 17:00

Maximal points: 200+10

http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

This is a 2 week homework and hence counts twice as much as the previous homeworks.

Submission instructions: Your submission for this assignment will include two parts. Submit both parts into your SVN directory into the according folder hw03. Keep in mind that we check if the code was copied from other people, so don't do it!

Part 1: Plots and answers: The first part will be a file that contains the required plots and answers to the questions. If you use other programs (such as MS-Word) to create your assignment, convert them to PDF (google for 'pdfcreator' for a free conversion program).

Part 2: Source code: The second part will consist of your source code. Follow the instructions below to create, name, and submit your source code. For all questions where you are asked to provide C code, we provide a corresponding C template file that you need to use to answer the question. In particular,

- do NOT change the signature of the functions
- do NOT change the type of global arguments
- comply with the given environment variables, do not add others
- do not cross-reference functions among your submitted files. Each .c file should be completely independent should be compilable on its own, (when compiled with our own main.c that you don't have access to).
- clean up your code as much as possible, do not leave in debug statements
- we provide a helpful sample main.c to show you how we would like the code to be structured. You are free to use it or not use it. It doesn't come with a timer, as you need to implement one. (possibly using the one from Assignment 2). Do not submit this sample file.

Verifying your code: All code that you produce as a part of this assignment (and future assignments too!) needs to be verified for computational correctness. For MMM, the easiest way is to compare to the standard triple loop (from assignment 2) for a few randomly selected input matrices. We will independently verify your code for correctness. Incorrect programs will not receive any credit.

Submitting your code: Your C files corresponding to the questions should be named code0.c, code1.c, code2.c, and code3.c. In all, you will submit the 4 code<n>.c files, and the finalcode.c file, plus any extra credit files. Do NOT change file names! Do not zip or otherwise archive the files.

1. *Miss rate (40 pts)* We consider the following program:

```
typedef double matrix[2][8]
double comp(matrix A) {
        int i;
        double t = 1000.0;
        for (i = 0; i < 7; i++) { // note: 7 not 8 because of the boundary
                t = t/(A[0][i] + A[0][i+1]);
                t = t/(A[1][i] + A[1][i+1]);
        }
        return t;
}
```

   We assume that a double requires 8 bytes, and that the array is cache aligned (that is A[0][0] is mapped to the first set and the first position in a cache block). Further, A has been initialized to contain only positive numbers. We assume a cold cache and ignore i and t in the cache analysis (they are held in registers). Recall that the miss rate is defined as $\frac{\#misses}{\#accesses}$. (Hint: It helps to draw cache and array.)

   (a) How many times is A accessed in this program?

   (b) The cache is direct mapped, has a size of 64 bytes, and 4 sets.

      i. How many doubles fit into one cache block?

      ii. What is the miss rate of the above program?

   (c) The cache is 2-way set associative, has a size of 128 bytes, and 4 sets.

      i. How many doubles fit into one cache block?

      ii. What is the miss rate of the above program?

2. *Reuse (20 pts)* Compute the reuse (as defined in class), both exact and in O-notation, for the following. Give enough detail so we know how you did it.

   (a) matrix-vector multiplication (MVM) in the form $y = Ax + y$, where x, y are vectors of length n and A is $n \times n$.

   (b) the scalar or dot product $\alpha = x * y = \sum_{i=0}^{n-1} x_i y_i$, where x, y are of length n.

3. *Blocking analysis for MVM (40 pts)* We consider double precision MVM in the form $y = Ax$, where x, y are of length n and A is $n \times n$. The goal is to perform a cache miss analysis with and without blocking analogous to the one done for MMM in class.

   Assume a cache size of C doubles and that C is much smaller than n. Assume a cache block size of 8 doubles and a cold (empty) cache.

   (a) With respect to locality, what is the fundamental difference between MMM and MVM?

   (b) Using the above assumptions, compute the number of cache misses of a standard double loop implementation based on

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

   Give enough detail so we know how you did it.

   (c) Now assume a blocked implementation based on dividing A into blocks of size $b \times b$, where 8 divides b and b divides n:

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

Compute the number of cache misses as a function of b. Give enough detail so we know how you did it.

(d) Continuing the previous question, derive the largest block size such that one block and two corresponding chunks of x and y fit into the cache (of size C). Show your work. What is the reduction in the number of cache misses for this block size?

4. *Mini-MMM (60 pts)* The goal of this exercise is to implement a fast mini-MMM (similar to how ATLAS generates 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) Use the code we provided in homework 2 that implements the MMM directly based on its definition (triple loop implementation), using the ijk loop order. Call this **code0**.

   (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. Manually unroll the innermost i- and j-loop and make sure you have alternate additions and multiplications (one operation per line of code). Perform scalar replacement on this unrolled code. Call this **code1**.

   (c) (Unrolling) Unroll the innermost k-loop by a factor of 2 and 4 ($K_U = 2, 4$, which doubles and quadruples the loop body) and again do scalar replacement. Assume that 4 divides N_B. This part gives you **code2** and **code3**.

   (d) (Performance plot, search for best block size N_B) Determine the L1 data cache size C (in doubles, i.e., 8B units) of your computer. Measure the performance (in Mflop/s) of your four codes for all N_B with $16 \le N_B \le min(80, \sqrt{C})$ with 4 dividing N_B. Create a plot with the x-axis showing N_B, and y-axis showing performance (so there are 4 lines in your plot for code0–code3). Discuss the plot. Also include answers to the following questions in your discussion: which N_B and which code yields the maximum performance? What is the percentage of peak performance in this case?

   (e) (Blocking for L2 cache) Consider now your L2 cache instead. What is its size (in doubles)? Can you improve the performance of your fastest code so far by further increasing the block size N_B to block for L2 cache instead? Answer through an appropriate experiment and performance plot.

5. *MMM (30 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. Discuss the plot.

6. *MMM (10 extra pts)* You can get up to 10 extra points by performing additional optimizations that further improve the performance of your code. Demonstrate with a suitable plot.

7. *Short project info (10 pts)* Submit (in Word .doc, .txt, or latex .tex) the following about your project:

   (a) An exact (as much as possible) problem specification for your class project.

   For example for MMM, one can be very precise, and 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 $C = AB \in \mathbb{R}^{n\times m}$.

   (b) The algorithm you plan to consider for the problem. You can actually sketch the algorithm or just give the name and a reference that explains it.

   (c) A very short explanation of what kind of code already exists and in which language it is written.