18-645/SP07: How to Write Fast Code
Assignment 3
Due Date: Thu Feb 14 6:00pm
http://www.ece.cmu.edu/~pueschel/teaching/18-645-CMU-spring08/course.html

Submission instructions: Your submission for this assignment will include two parts.

Part 1a: 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). Name your file '18645-assign3-userid.pdf' where 'userid' is your andrew user id. The .pdf file must include all plots and figures. Do not put the .pdf file in a zip or tar archive - attach it separately. Send it along with part 1b (see below) to <schellap+18645-assign3@andrew.cmu.edu>. *In addition to the electronic copy, you must also submit a print-out of your pdf to the TAs at PH-B10 or to Carol Patterson at PH-B15.*

Part 1b: Class project problem specification (See Question 5): Email your *source* files (MS-Word .doc, .txt, or latex .tex) file to the email address shown above. Also create a print-out and drop it off along with the rest of your assignment.

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 Assigment 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. Extra credit programs should be named codeX1.c, codeX2.c etc. 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! Ensure that you have your name (commented out) on the first line of each source file. Do not zip or otherwise archive the files.

Place your files in the following AFS directory (already created for you):

/afs/ece/class/ece645/submit/assign3/<andrewid>/

Replace <andrewid> with your andrew (same as ECE) user id. You can access this directory using any cluster machine, or with a Windows AFS client. (The permissions on the AFS directory require you to be logged in with your ECE AFS credentials, which is automatically done when you login to any ECE cluster machine.

1. *(65 pts)* This exercise builds on lecture 7. The goal of this exercise is to implement a fast mini-MMM 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 Assignment 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 $16 \leq N_B \leq min(80, \sqrt{C})$ 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) (Loop order) Does it improve if in the best code so far you switch the outermost loop order from ijk to jik? Create a plot to show the answer.

   (f) (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.

2. *(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 an interesting range of sizes n (up to sizes where the matrices do not fit into the L2 cache). x-axis shows n; y-axis performance in Mflops. Discuss the plot.

3. *(Extra credit, up to 20 pts)* If you went through all steps in 1 and 2, you'll receive extra points for any other optimizations or interesting experiments that you perform. Examples include skewing, buffering, software pipelining, or anything else you can think of. Name the corresponding codes **codeX1**, **codeX2**, ... Create appropriate plots with brief discussions.

4. *(5 pts)* How many hours did you spend on this assigment?

5. Submit (in Word .doc, .txt, or latex .tex) an exact (as much as possible) problem specification for your class project as explained in the second lecture.

   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 $C = AB \in \mathbb{R}^{n \times m}$.