18-645/SP07: How to Write Fast Code
Assignment 4
Due Date: Thu Feb 21 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 1: Writeup. The first part will be a file that contains a writeup. 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-assign4-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-assign4@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 2: Source code. The second part will consist of your source code. Since this will vary for different projects, there is no single format. Most likely, you will have your implementation, and a timer and a verifier that work on your implementation.

Place all your files (including timer, verifier, Makefiles etc. - do not archive/zip them) in the following AFS directory (already created for you):

/afs/ece/class/ece645/submit/assign4/<andrewid1>-<andrewid2>. . . /

Where each <andrewid> is the andrewid of one of a group member. Refer to Homework 2 for more information about copying files to AFS.

This week, you will get your research project started, i.e., there is no homework. Instead, you should work with your project partner(s). Last week, you created the problem specification for the problem you will implement in your project.

1. Create a straightforward C implementation of your problem. Straightforward means the way you would do it without having optimizations in mind. In most cases this means a direct implementation of a known, good algorithm. Verify the code and explain how you verified.

   This implementation will serve as a baseline for future optimizations and you can use it for verifying future versions.

2. Determine a suitable cost measure and then the cost for your implementation. In most cases this is the arithmetic cost (floating point adds and mults). Ideally, you will determine the cost (or at least the highest order term) precisely by analysis, or by measuring if this is not possible. The cost is parameterized by at least one parameter, usually the input size n, but there could be more parameters.

3. Measure the runtime of your implementation for a range of input sizes n and compute the performance. Create a performance plot with n as the x-axis. What percentage of the scalar (no vector instructions, no multithreading) peak performance on your machine do you achieve?

Depending on the project, the above tasks may be very easy or already challenging. If it is very easy more start thinking about, and trying some optimizations, and submit the results.