263-2300-00: How To Write Fast Numerical Code
Assignment 4
Due Date: Thu April 14 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

In these 2 weeks you will start working on your research project in the teams listed on the course website. Hence there are no homework points since the overall project itself counts as 40% of the total grade. For all submissions including the current status of your code you will use the project svn directory created for you (which is different from the svn directory you used for the first 3 homeworks - you should have received a mail with the project SVN link). Feel free to structure the directory as you see fit except for the specific instructions below. I do understand that you will also spend some time preparing for the exam.

The project svn is different from the homework svn.

The exact procedure will differ from project to project. Below we provide a generic strategy that you can follow where you see fit. We will check progress after the due date above. Further, one or two lectures may be replaced by one-on-one meetings with the instructor to discuss the status and possible directions.

For the due date above you have to submit the following to the project svn:

1. In the root directory a document called initial.pdf that contains your answer to question 7 of the last homework (sorry for the copy-paste).
2. The current status of your code should be maintained in a subdirectory called code/
3. In the root directory a document called report1.pdf where you briefly (e.g., half a page or one page text) describe what you did during this period. You are encouraged to include relevant graphics if available, which then should be briefly explained. Include compiler, version, and flags were applicable. Don't be overly exhaustive. Just provide the crucial information.

Outline of the steps in the (entire, not just these 2 weeks) research project. Not all may apply and additional ones may apply:

1. Create an initial C implementation as you would usually have done.
2. Create an automatic timing and verification infrastructure since you will create several versions during this project.
3. Perform a cost analysis and create a performance plot. Choose as wide a range of input or input sizes as feasible (meaning until the computation takes several seconds). Determine the percentage of peak performance. At this stage you should disable vectorization but play a bit with optimization flags to find good choices. In performance results always include compiler, version, and flags.
4. If your algorithm consists of several steps, profile it, i.e., determine the runtime contribution of each part so that then you can focus on the bottleneck.
5. Optimize the bottleneck using techniques from class. Pretty much everything that improves runtime is allowed. You will create many versions. Include the most important ones in your performance plot. You may need to reprofile as the bottleneck may change.
6. Start with higher level optimization such as restructuring for cache efficiency if possible. Basic block optimizations should be only done afterwards.
7. You may want to change intermediate data structures for efficiency. Arrays are often (but not always) most efficient. The interface of the function should typically not undergo major changes.
8. Any form of analysis (reuse, locality etc.) that can be done should be done and briefly reported and adds value to the project.

9. Once you are convinced that nothing can be improved anymore, start with SSE. Feel free to try compiler vectorization (the Intel manual will help). Repeat the above steps as often as needed.

10. If there is the chance to develop a small generator for a part of the code that's of course great.