CPSC 3300

Project 2 - Loop Optimizations

Due date: 11:59 PM, April 1 on Canvas

**Grading Weights:**

48% description of the techniques

4% description of loop-carried dependencies

4% description of aliasing

30% performance improvement examples

10% bibliographic references and citation style

4% section headers

This is an individual or team of two project assignment.

**Summary of the assignment**

You will write a 3 to 5 page paper (6 to 10 pages for teams of two) to

1. describe the loop optimization techniques listed below (at least 3 working individually or at least 6 as teams of two). You shall use example codes to show the transformations before and after the optimization,
2. describe two hurdles to loop optimization: loop-carried dependencies and aliases, and
3. run experiments and report on performance benefits obtained by applying loop optimization techniques available in gcc or g++ to the whetstone benchmark.

**Details:**

**List of loop optimization techniques:**

* implicit threading of loops using OpenMP
* loop fission
* loop fusion
* loop interchange
* loop invariant code motion
* loop peeling
* loop strip mining
* loop tiling/blocking
* loop unrolling
* loop unroll and jam
* loop unswitching
* software pipelining

**Loop-carried dependencies**

[definition and how these dependencies affect the ability to optimize loops]

**Array element aliasing**

[definition and how these aliases affect the ability to optimize loops]

Be sure to provide citations for any source you use. (Don't rely entirely on Wikipedia for the descriptions; double check with other sources.)

**Hands-on test and results report for GNU compilers on the Whetstone code.**

There are many techniques available in GNU compilers (gcc, g++, or gfortran) and they can impact the performance of programs, e.g., the Whetstone code. Use command man gcc to get the manual.

Note that the compilers have several levels of optimizations, i.e., -O0, -O1, -O2, etc. The higher the number, the more optimization techniques are applied. These levels directly control some optimizations, but not all. Again read the gcc manual. On the other hand, most optimizations are only enabled if an -O level is set on the command line. Otherwise they are disabled, even if individual optimization flags are specified. For example, if you would like to turn on  -funroll-loops, you can use gcc -O2 -funroll-loops. You can also refer to this online page <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>.

Start with the Whetstone code and test the various compiler options. (I would recommend that you write a Makefile to compile the program and a script to run the experiments.)

Report the hardware/OS platform and the version of gcc/g++ you use for experiments; also give the exact compiler invocation flags for each test for Whetstone looping for 200,000 times. Report the execution time in a table with each optimization technique enabled.