Before the Lab

Read 4.1 and 4.4 from
https://cnx.org/contents/u4IVVH92@5.2:bEZZukPR@1/Introduction-to-the-Connexions-Edition
(Severance and Dowd). This is a classic text on optimization--and while some of the hardware examples are out of date as the text is now out of print, the theoretical descriptions of loop optimizations are all still valid.

Lab Activity

This activity will allow you to run a series of codes that test principles of loop optimization. Each code will allow you to compare loop performance on a loop that is, or is not, designed based on optimization principles.

For the code examples, they can be run either by downloading the files and compiling them on the machine of your choice (you should have a C/C++ compiler on the machine for which you are able to specify command line options, and can produce code to run at the command line.) Alternately, a Google CoLab notebook is provided where you can run each of the codes on the underlying virtual machine, by running first the cell to write the code to a file on the machine, then the following cells to compile and run the code.

Running example of the examples will consist of executing "./*example_name* 0" or "./*example_name* 1", where *example_name* is the example to be run. The unitstride.c example is the one exception, where you would run "./unitstride *stridelength*" where *stridelength* is a large integer.

1. Of the following optimization concerns, rank them in an order you feel is least important to most important: redundancies within loops, use of transcendental functions when simple math functions will suffice, function overhead, condition overhead, out of order memory access.
2. Run each of the provided examples. Classify each of the examples according to the categories in question 1 (is it an overhead question, an out of order access question, etc.) Describe how changing the input to the code changes performance, and discuss the degree to which performance is increased.
    a. arraystride.c allows you to perform 2D array access either by rapidly changing the first index or the second.
        i. Mode 0, stride by inner (rapidly changing) index
        ii. Mode 1, stride by outer (slowly changing) index

b. unitstride.c operates on a 1-D array either sequentially, or striding a number of steps at a time. (Note unitstride.c differs slightly in usage in that if equivalent, you would expect striding through the sme array in steps of 2 would take half the time, etc.)
    i. Provide a stride length, and the code will operate on an array either 1 item at a time in order, or jumping through the array with the stride length you have provided.
c. loopcondition.c has a conditional behaviour on the first and last item of the loop, and allows you to perform the loop with either separate conditions outside of a smaller loop, or with the conditions inside the loop.
    i. Mode 0 places the condition outside of the loop
    ii. Mode 1 places the condition inside of the loop
d. loopinvariantcode.c has a redundant statement inside of a loop that can be moved outside of the loop.
    i. Mode 0 places loop invariant code inside the loop.
    ii. Mode 1 places loop invariant code outside the loop.
e. strengthred.c replaces a call to pow(x,2) with x*x.
    i. Mode 0 uses pow(x,2)
    ii. Mode 1 uses x*x
f. inlining.c allows you to compare a small function written as a function or as a #define statement.
    i. Mode 0 calls a function
    ii. Mode 1 calls a #define statement macro instead of a function
g. arrayfunc.c allows you to explore the performance difference of passing an array to a function to be looped over, versus creating a loop to call a function on array elements.
    i. Mode 0 loops over an array and applies a function to each element of an array
    ii. Mode 1 passes an array to a function, which loops over the array

3. If running these exercises changes your initial ranking of importance, re-rank and explain why. Describe which of these items you feel an automated compiler might be able to spot and fix, and which can only be changed by the developer. Feel free to re-run each of them with optimizations turned on instead of off to see if that changes performance.
4. Based on your exercise, state what you think is the single most important factor in loop optimization and why.