Meeting Notes and Assignment 6/19/19

Paul Scanlon and Matt Murdoch

Colorado State University, Research for Dr. Sanjay Rajopadhye, Summer 2019

We began by explaining how we fixed the self dependence in the LUD algorithm. The solution was implemented by restricting the body of the summation. Inspiration for this fix came from construction of the Shadows algorithm.

Next we showed the algorithm implemented for the shadows problem and explained the variables. Knowing that this algorithm works but with poor run-time, we should now pursue optimization.

A Benchmark of one second is a good standard for excluding noise produced by external inaccuracies. Once a problem size is determined that supports this run-time, improvements can by made without the worry of measuring noise. Anything less than one second is typically noise. For the Shadows problem, the inaccuracy of the timer can reach tenths of seconds, so problem sizes that run for more than one second are desired.

Zero error in terms of run-time measurement refers to the average time it takes to run a program that should take zero time. The results indicate the finest accuracy that should be reported as to avoid noise in the result. Zero error can be implemented with two calls to the timer in succession.

Assignment

Study the wrapper, specifically the $if\ defined$ sections. Look for compiler flags that are not implemented in the $makefile$ and implement them to unlock more verification and checking versatility. The $if\ defined$ statements can be reflected in the $makefile$ that have not been included yet. Include them all. There are flags for checking, no prompt, timing, etc. Figure out what happens for each flag and embellish the $makefile$ after understanding them.

Looking at the C functions created, the contents of Shadows.c and Shadows-wrapper.c, optimization potential becomes more evident.

Shadows_verify.o is created when $make\ verify$ is called. The $verify$ function expects Shadows.c which corresponds to the system code. The *malloc* for all of the variables is done in the wrapper. Shadows.c calls the wrapper to use the variables and print the answers.

When calling $make\ verify$, some of the function may not work because we have done minimal debugging. We want to check accuracy for a large range of inputs and expected outputs now that manual debugging is complete. Once the code is verified as correct, go into the C code produced and take the function it has made and save it as a new verify script. Once the Shadows_verify.c script is created, that is the gold standard. Now you have the key to test optimization strategies for accuracy with. Modifying the alpha script will produce a new Shadows.c that can be checked against the previously made Shadows_verify script.

Find out how to time the program with the *makefile*. Gather data empirically and plot it as a function of N, the problem size. Then, check if the program is running in quadratic time. How can you improve the algorithm and does the improvement affect worst case run-time or special cases? By plotting empirical data, we an find statistics about the performance of the algorithm.

Storing the empirical data can be done by redirecting the wrapper output to a file. The flag $-DRANDOM$ produces a proper repeatable experiment with different random values each time so long as it is not seeded.

In order to facilitate this, we should read about *makefiles* in Zybooks or on the CS253 homepage from last semester.

Optimization Questions / Direction

If we know information about i (height i), what can we predict about i+1?

How is i affected by i-1?

How does i-1 affect i+1?

These three questions will guide an optimization of a stencil like system of checks that reduces the number of trigonometric evaluations that need to be done by remembering the state of consecutive shadow locations.