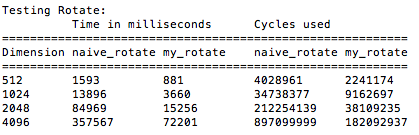
CSCI 2461 Project 5 Report

The Rotate Function

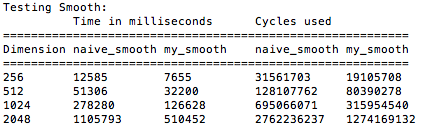
1. Optimization techniques – Using the code-optimizing transformations learned in class, the rotate function greatly improved in performance. The two main optimization techniques used was loop fusion and unfolding.
2. Reasoning – Multiple for loops for fused together when the array dst was equated to the array src. Thus, the number of overall iterations that the program ran through was less, and took up less overall memory. When the unfolding procedure was carried out the two variables became invariant variables in the resulting loop. The dynamic execution of the loop improved overall run time.
3. Performance Results –



1. Specific Techniques – After running the code and obtaining the time taken between the techniques it was found that the unfolding transformation was more efficient than the loop fusion technique, although both transformations sped up the time taken by a considerable margin.

The Smooth Function

1. Optimization techniques – In the smooth function, the major optimization techniques tried were in-lining, code motion and loop interchange.
2. Reasoning – First, the accumulate\_sum function was moved into the my\_smooth function to save an extra function from being used. Moreover, the function becomes localized throughout with the exception of the static max() and min() functions in the for loops. With this change, there are no longer any calls within the function my\_smooth(), which reflects good in-lining and localization. Attempted to optimize the solution through reducing the memory access also helped as the pointer sum\_num would not continuously have to be accessed with every loop iteration. However, reducing the memory access did not prove to be as effective as expected. Finally, adjusting pixel *p* outside of the for loop displayed code motion (specifically loop-invariant code motion).
3. Performance Results –



1. Specific Techniques – After compiling the code, it was found that the in-lining transformation was the most effective, although code motion sped up the process as well. The loop interchange was fairly ineffective and was not used in the final implementation of executing the smooth function.