COSC561 – M6.1 Programming Assignment 4 - cpass

Giuseppe Schintu

University of Tennessee, Knoxville

COSC561 – M6.1 Programming Assignment 4 - cpass

With some previous experience on LLVM acquired during the CSEM project, I was aware of the LLVM API and high-level configuration and potential uses. I also got the gist of the Muchnick Copy Propagation Algorithm and DFA analysis. However, for some reason, I spent a few days studying about the LLVM IR Instructions, BitVector, and Iterations. I was able to find some distant related code that worked on Machine Copy Propagation, and with that I got an idea of what commands and API instructions I could use. I also researched about how LLVM understands the Store, Load, Other instructions, and how the instructions can use same memory location, which ultimately can be optimized to remove some redundant instructions.

As suggested during class meetings, I started working on the localCopyPropagation, but the instructions suggested a straightforward iteration on the blocks and a call to propagateCopies. I got distracted with the ACPTable and worked on figuring out how to populate the DFA as shown by the REF output. Soon, I was going all over the place attempting to implement initCOPYAndKILLSets() and initCPInAndCPOutSets(). After a few days of struggle in trying to remove load instructions from propagateCopies, I thought I was optimizing the code by removing “duplicate” instructions. However, after getting some help from the professor and TA, I realized I was removing load instructions in the wrong order, thus altering the logic of original code.

Perhaps one of the biggest struggles in this project was to find LLVM instructions that could perform operations instead of resorting to iterate from copy_idx, idx_copy and use copies and nr_copies range to extract values. However, some of the interesting instructions to replace getOperand(0), and getOperand(1) where from the store pointers(getPointerOperand() and getPointerValue) and it seemed like they made the code more readable in some routines.

Additionally, I found the use of replaceAllUsesWith() which I've learned it has to be used in the proper order and context, or it will replace load instructions that end up altering the original code.

Finally, the most tedious part was getting the initCPInAndCPOutSets() and initCOPYAndKILLSets() to work correctly for the most part, mainly because from my understanding CPOut needs the COPY and the KILL to work correctly. The Muchnick DFA was a bit difficult to ensure I was not missing concepts, but the algorithm itself was relatively reasonable. However, the implementation and calibration between KILL.set(), and the impact on CPIn and CPOut was tricky to get it right. So, it seems like I missed some extra optimization in some blocks.

In conclusion, this was an interesting project. It certainly emphasizes the importance and complex work that compilers do, and this project in particular gave me a new perspective towards the computer scientists that are continuously researching and figuring out better ways to optimize intermediate code.