CS 4390/5390 Fall 2025
Advanced Compilers
Shirley Moore, Instructor

Implementing Loop Optimization Passes in LLVM

For this assignment, we use the LLVM tutorial at https://github.com/banach-space/llvm-tutor as a starting point and extend it with additional passes. Following the notes at https://www.cs.cornell.edu/courses/cs6120/2025sp/lesson/8/ , we will implement basic versions of loop invariant code motion (LICM) and induction variable elimination (IVE). You may discuss this assignment with each other, but each of you should write your own code.

1. Simple LICM
   You are provided with starter code in the attached SimpleLICM.cpp. To finish the pass, implement the worklist algorithm to determine if an instruction is loop invariant. Focus only on register-to-register computations -- i.e., do not hoist instructions that read or write memory, and do not hoist phi instructions. Do not use the LLVM `isLoopInvariant()` function. When you are done, you will have a basic pass for detecting and hoisting loop-invariant instructions. For example,

   shirleymoore@Shirleys-MacBook-Pro build % opt -load-pass-plugin ./lib/libSimpleLICM.dylib -passes=simple-licm -S -o outputs/matmul_licm.ll inputs/matmul_canonical.ll
   Hoisting: %7 = sext i32 %.037 to i64
   Hoisting: %8 = getelementptr inbounds [800 x double], ptr %0, i64 %6
   Hoisting: %14 = sext i32 %.026 to i64
   Hoisting: %6 = sext i32 %.037 to i64
   Hoisting: %7 = getelementptr inbounds [800 x double], ptr %0, i64 %5
   Hoisting: %22 = sext i32 %.037 to i64
   Hoisting: %23 = getelementptr inbounds [800 x double], ptr %2, i64 %7
   Hoisting: %8 = sext i32 %.013 to i64
   Hoisting: %9 = getelementptr inbounds [800 x [800 x double]], ptr %1, i64 0, i64 %5
   Hoisting: %14 = sext i32 %.013 to i64
   Hoisting: %15 = getelementptr inbounds [800 x [800 x double]], ptr %2, i64 0, i64 %7

2. Induction variable elimination (IVE)
   You are given two example codes AffineRecurrence.cpp and DerivedInductionVar.cpp that implement affine-recurrence and derived-iv analysis passes, respectively. These

passes use the LLVM ScalarEvolution class. Study and run these passes so that you understand them. Then extend the DerivedInductionVar.cpp code as follows:

a. Extend DerivedInductionVar.cpp to identify induction variables in the inner loops of loop nests.
b. Extend the pass into a transformation pass called that actually eliminates the induction variables.

How to submit: Please implement and test your code in the llvm-tutor infrastructure so that the instructor can use that infrastructure to build and test your pass. The best way to do this would be to turn in a link to a github repository containing your llvm-tutor setup that the instructor can clone or update (if already cloned from a previous assignment).

Grading: 50 points for each pass, with 40 points for correctly working code and 10 points for README and test cases. As part of testing your passes, be sure to test your optimized IR code to make sure it is still correct and produces the same results as before optimization.