CS 6241 : Compiler Optimizations
Homework II
Points : 100

Start: Feb 21st Due : March 3rd, 2014

Important policies:

1. Georgia Tech honor code will be strictly enforced for this and other projects.
2. This is an individual assignment. You can discuss your ideas but should work individually.
3. Copying of code or solutions from any source is an act of plagiarism.
4. Make reasonable assumptions and clearly state the assumptions made.
5. Please use Piazza for any questions or clarifications.

Please Note: Turn off all inbuilt optimizations when you are generating LLVM bytecode.

Part 1(Points: 45)

Reaching definition (RD) analysis is a data-flow analysis which determines the definitions which reach each program point. Ignoring *phi* functions in LLVM bytecode for now, complete the following:

(i) Implement the naive reaching definition analysis. (5 points)
(ii) Implement dependence-based reaching definition analysis and compare the efficiency of your implementation to the naive analysis. (20 points)
(iii) Implement demand-driven reaching definition analysis for loops and compare the efficiency with your previous implementations. (20 points)

Part 2(Points: 20)

(i) *Constant Propagation* is an optimization implemented using forward data flow analysis. It eliminates redundant computation of expressions which evaluate to constants. Implement a constant propagation pass (transform pass) and record the effectiveness of you implementation. Note the number of expressions replaced in your test code. (10 points)
(ii) *Unused variables* are variables which are defined but never used. Such variables are generally optimized out as part of link-time optimization(LTO). We want to do it beforehand using our pass framework. Implement a pass which detects and removes all unused variables in a program.

Part 3(Points: 35)

Data-flow analysis can be used to detect unsound programs that have un-initialized variables. An *uninitialized* variable is the one which is used at least one program point before it is defined. Since garbage values can reside in uninitialized variables, it is desirable to detect them. An uninitialized variable can propagate such values to other variables' definitions that are derived from them and this can continue transitively. We call such a set of definitions as *unsound.* Implement an analysis pass to detect and warn about all unsound definitions in a program. You need to map the IR variables back to the source code and report them.

Deliverables:

1. A report containing the above results.
2. Source code for the implemented passes. Please ensure that your source code is documented by appropriate use of comments.
3. Test code using which you verified your passes.