The Project

Description

For the class project, you will extend a simple compiler with optimizations as part of a team of 3 to 4 people. This will provide you with concrete experience in engineering the techniques described in the lectures and readings.

We will use the Tiger compiler from the textbook, which is written in Java. We're starting from the solutions to the undergraduate project, namely a complete but unoptimizing compiler for the Tiger language, generating MIPS assembly code. There are a couple of test cases; you'll want to write a bunch more to help you debug your compiler.

The first task is to design and implement a simple intermediate representation for your compiler. The existing compiler produces a singly-linked list of basic blocks, each of which holds a singly-linked list of simple intermediate instructions. You will want to devise your own intermediate representation and write converters to your representation from the existing one, and back.

The second task is to design and implement a general framework for doing arbitrary forward or backward dataflow analyses over your intermediate representation. The framework should allow dataflow analyses to be defined in lattice-theoretic terms, and it should automatically handle the process of applying flow functions and meet operators, manage iteration, and report analysis results back to clients. It must handle irreducible control flow graphs. It need not support automatically composing analyses or transformations (like Vortex's framework), but it should support writing a single analysis with little trouble.

The third task is to write several analyses and optimizations. Intraprocedural optimizations at least will benefit from using your framework.

Requirements

All projects should perform the following optimizations:

- peephole optimizations:
  - constant folding (including branch conditions and eliminating unreachable code)
  - arithmetic simplifications: multiplies to shifts
- local optimizations:
  - instruction scheduling (to avoid load stalls)
- intraprocedural optimizations:
  - constant propagation
  - common subexpression elimination or pointer analysis
  - loop-invariant code motion or induction variable elimination
  - dead assignment elimination
- interprocedural optimizations:
  - inlining

Written Reports

Along with the code of the compiler, a report must be written describing the overall organization, the intermediate representation, the interface and algorithms in the dataflow analysis framework, lattice-theoretic specifications of the intraprocedural analyses, and a brief description of the implementation strategy of the other optimizations. This design document should be written incrementally as the compiler is implemented, and will be reviewed as part of the checkpoint dates. The report should be no longer than 15 pages when completed.

Time Schedule and “Deliverables”

Several dates are important for completing the project:

1/28: Project Teams Formed - turn in list of members of each team.

2/6: Intermediate Representation - the compiler should build and generate code from an appropriate intermediate representation; no optimizations necessary. A draft of the project report describing the IR should be turned in, too.

2/18: Data Flow Analysis Framework - the compiler should support a general data flow analysis framework, with constant propagation & folding and dead assignment elimination working using the framework. An extended draft of the project report also describing the analysis framework should be turned in, too.

3/9: Completed Optimizer - the compiler should be augmented with all remaining optimizations.

3/13: Final Project Reports Due - the final version of the report should be completed.

No late projects will be accepted. Projects and reports are due at the beginning of class.