Thoughts on the Lab

Copyright 2015, Keith D. Cooper & Linda Torczon, all rights reserved.

Students enrolled in Comp 512 at Rice University have explicit permission to make copies of these materials for their personal use.

Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.

Citation numbers refer to entries in the EaC2e bibliography.
One Big Question

One of the critical design decisions in the lab is choice of optimizations

• Intent was for you to get started early and experiment
  ♦ Selection driven by problem areas in the benchmark codes ...
  ♦ Too late for that idea ...

• We have seen several transformations that might be relevant
  ♦ Value Numbering (LVN, SVN, DVNT)
  ♦ GCSE via AVAIL information
  ♦ Constant propagation: Kildall’s algorithm, SCP, and SCCP
  ♦ DEAD and CLEAN
  ♦ Loop-invariant code motion: both naïve algorithm from lecture 11 and LCM
  ♦ Loop-unrolling
  ♦ Strength reduction: both Cocke-Kennedy technique and OSR

Some opts cannot help on the lab, such as profile-guided code positioning or inline substitution.
Second Big Question

Some of those techniques require SSA form

• Transforming the code into SSA is a major undertaking
  ♦ Game theory: which way do you spend your time?
  ♦ Early in semester, I would have said “build SSA”
  ♦ Late in semester, I would say “build something that works”

What effects are important in the execution model?

• Reducing the overall operation count
  ♦ Redundancy elimination, constant propagation, useless and unreachable code elimination, code motion

• Replacing expensive operations with less expensive ones
  ♦ Constant propagation and strength reduction (both weak form & strong form)

• Hiding latency
  ♦ Instruction scheduling
Second Big Question

What would I do (at this point in the semester)?

• Reducing the overall operation count
  ♦ Redundancy elimination (LVN or SVN, DVNT really needs SSA form)
    → Wouldn’t use GCSE with AVAIL because I want constant folding & algebraic simplification
  ♦ Constant propagation (I would count on LVN or SVN here)
    → The easy to implement algorithms require SSA
  ♦ Useless and unreachable code elimination (DEAD and CLEAN)
  ♦ Code motion: I would try one of the simple algorithms
  ♦ Loop Unrolling: Easy to do, not sure how much it will help

• Replacing expensive operations with less expensive ones
  ♦ Strength reduction
    → With SSA, I would do OSR
    → Without SSA, I would try an ad-hoc approach to weak strength reduction, maybe working it into LVN

• Hiding latency
  ♦ Instruction scheduling

The tradeoffs will depend on what code is in the benchmarks. Lots of loads & stores? Lots of multiplies?