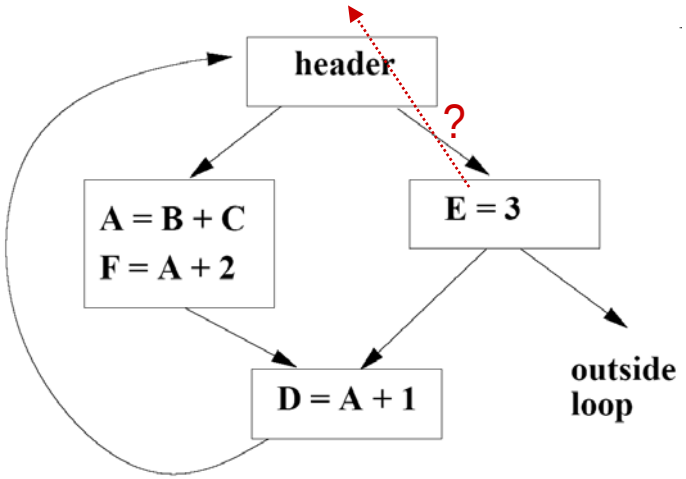
Loop Invariant Computation and Code Motion

- Loop-invariant computation
- Algorithms for code motion
- Summary: 13.1.2 (global CSE), 13.2

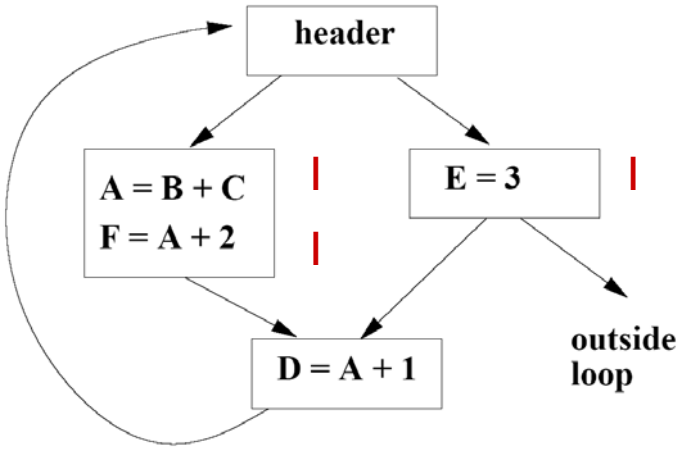
Loop-Invariant Computation and Code Motion

- Loop invariant computation
  - A computation whose value does not change as long as control stays within the loop
- Loop invariant code motion (LICM)
  - Move a loop invariant statement within a loop to the pre-header of the loop



Example

- Which statement is a loop-invariant?
- Which statement can we move to the preheader?



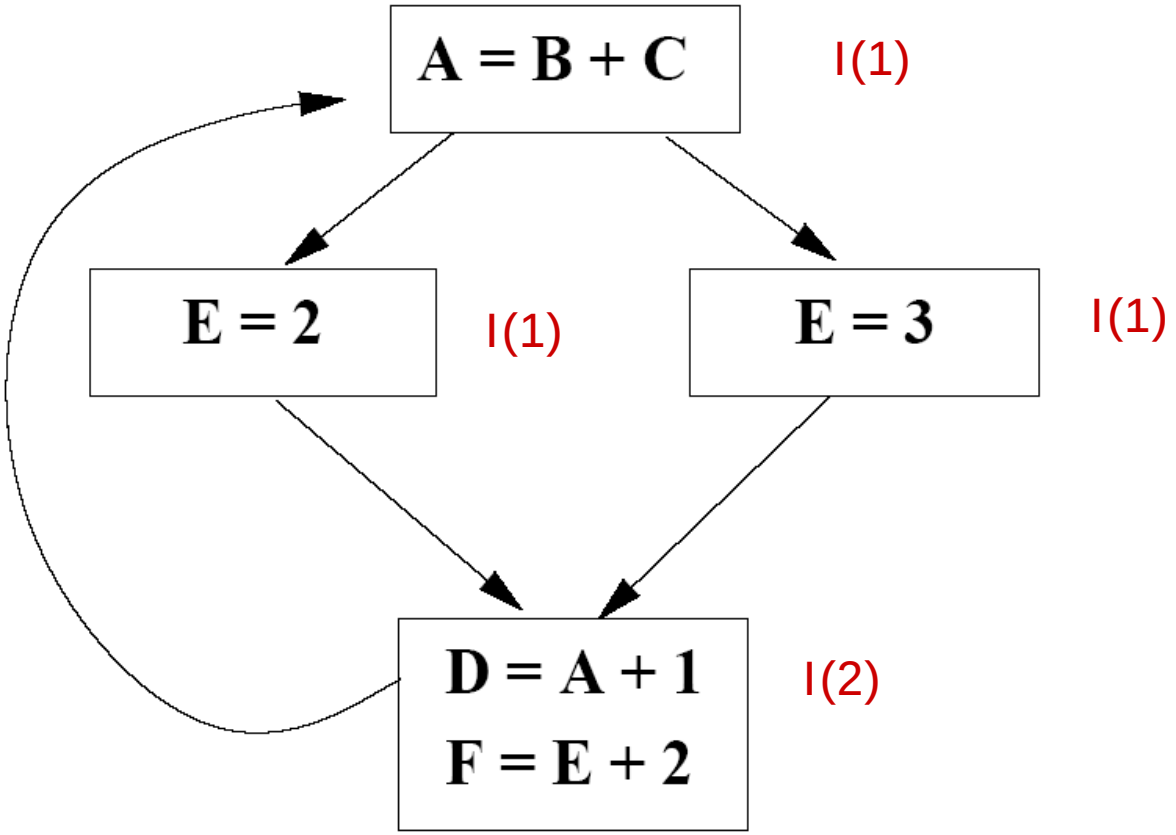
LICM Algorithm

- Observations
  - Loop invariant: operands are defined outside loop or are defined by loop invariants
  - Code motion: not all invariant statements can be moved to the pre-header
- Algorithm
  - Detect loop invariant computations
  - Check conditions for code motion
  - Code transformation

Detecting loop invariant computation

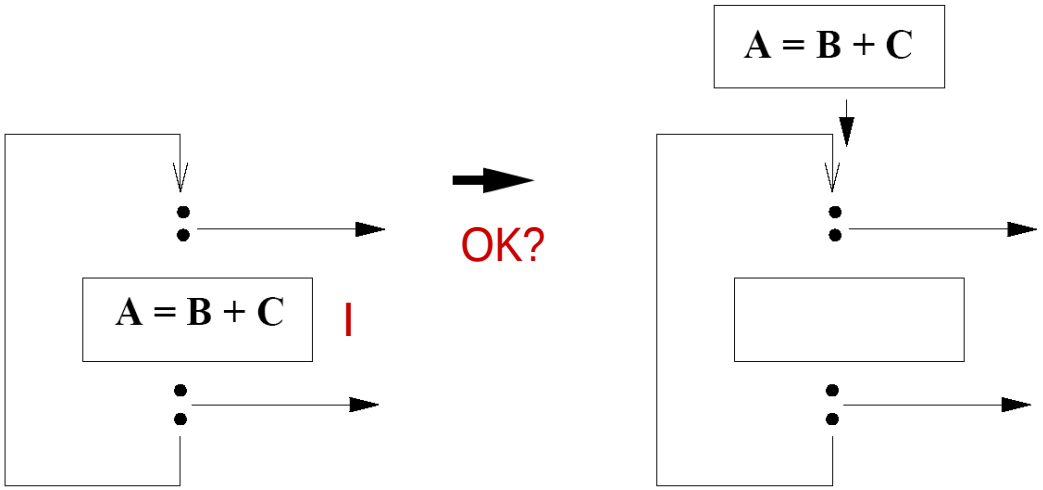
- Compute reaching definitions
- Repeat: mark $A=B+C$ as invariant if
  - All reaching definitions of B are outside of the loop, or there is exactly one reaching definition for B and it is from a loop-invariant statement inside the loop
  - Check similarly for C
- Until no changes to the set of loop-invariant statements occur

Example



Conditions for Code Motion

- Correctness: movement does not change the program semantics
- Performance: code should not be slowed down



Need information on

- Control flow of the loop: dominate all the exits
- Other definitions: no other definitions of A
- Other uses: all uses of A are dominated by block b

Code Motion Algorithm

Given a set of nodes in a loop

- Compute reaching definitions
- Compute loop invariant computation
- Compute dominators
- Find the exits of the loop, which are nodes whose successors are located outside loop

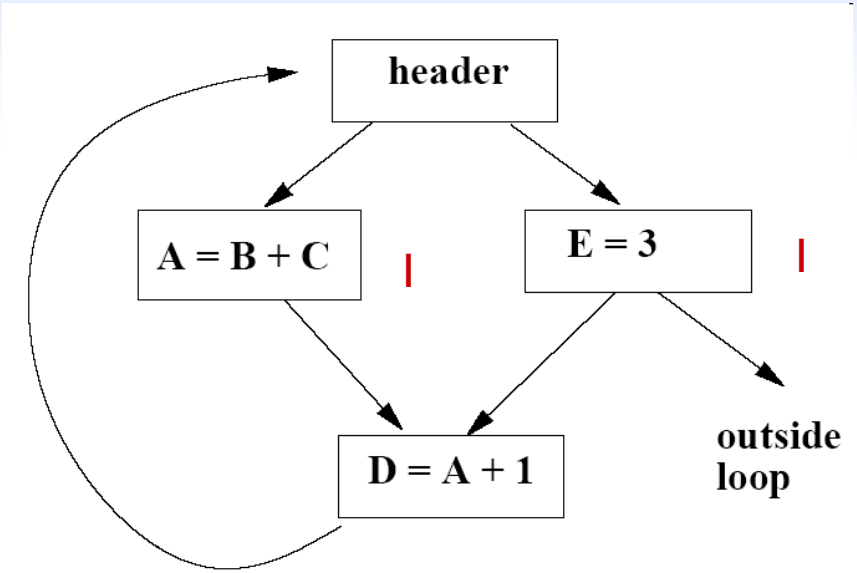
Code Motion Details

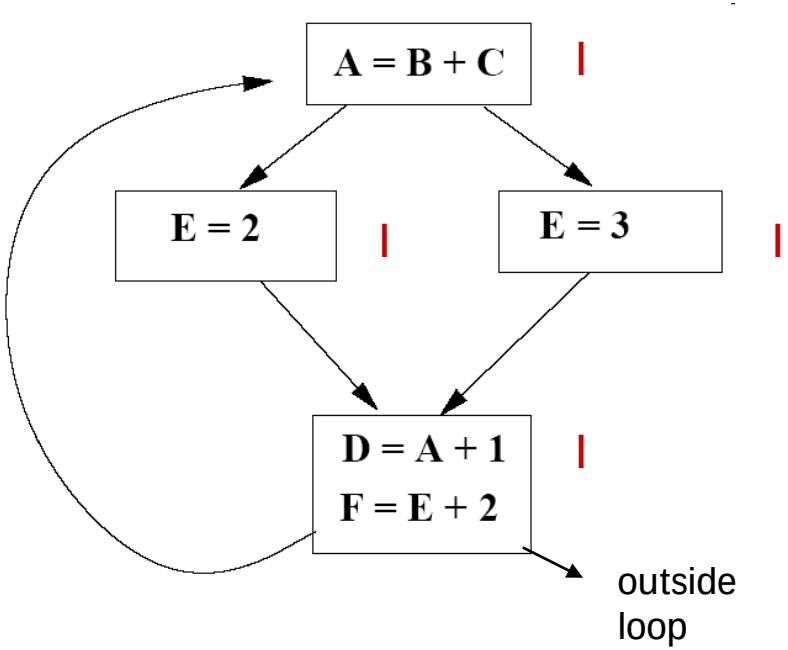
- Candidate statement for code motion
  - Loop invariant
  - Located in a block that dominates all the exits of the loop
  - Assign to a variable not assigned to elsewhere in the loop
  - Located in a block that dominate all blocks in the loop that contain the use of the variable

- Visit blocks in a reverse-postorder
  - Move the candidate statement to pre-header

  if all the invariant operations it depends on have been moved

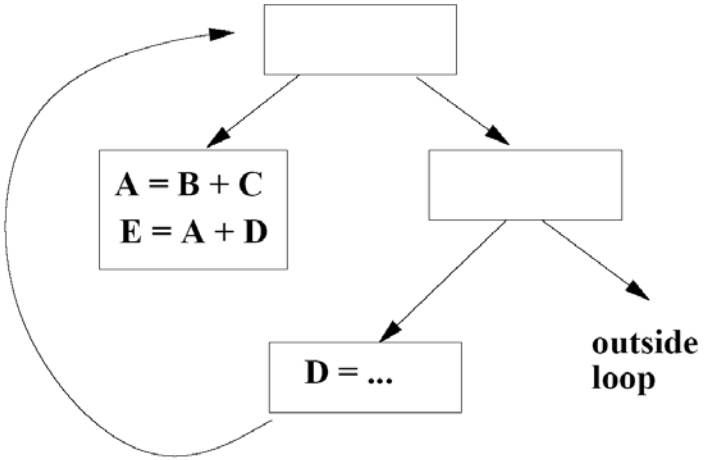
Examples





More Aggressive Optimizations

- Gamble on: most loops get executed
  - Can we relax constraint of dominating all exits?
    - If liveness check and exception check is satisfied



Landing Pads

Code for most loops is generated in a test-repeat-until form to simplify dominance

Landing pads: