Loop Optimizations

Why loops?

- Loop Constructs
  - while …
  - do ... while ...
  - for ...
- Why are loops important?
  - 90/10 rule
  - 90% of execution time, 10% of code (loop)

Agenda

- Low level loop optimizations
  - Code motion
  - Strength reduction
  - Unrolling
- High level loop optimizations
  - Loop fusion
  - Loop interchange
  - Loop tiling

Loop optimization

- Low level optimization
  - Moving code around in a single loop
  - Examples: loop invariant code motion, strength reduction, loop unrolling
- High level optimization
  - Restructuring loops, often affects multiple loops
  - Examples: loop fusion, loop interchange, loop tiling

Low level loop optimizations

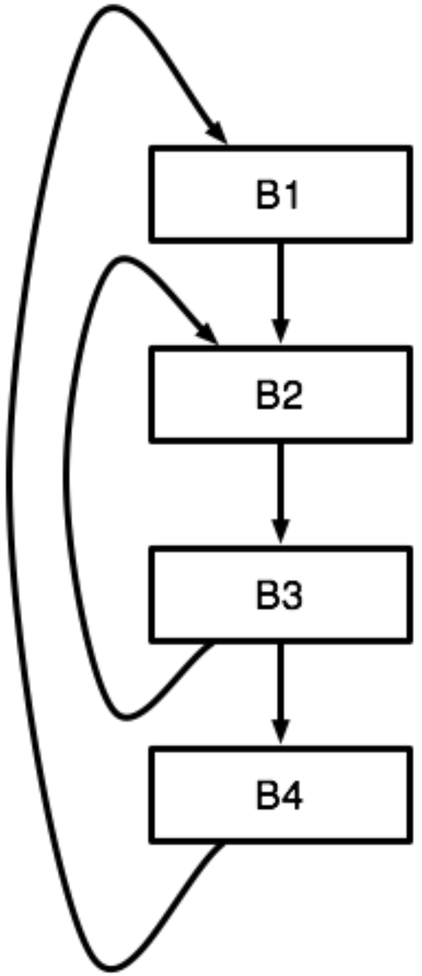
- Affect a single loop
- Usually performed at three-address code stage or later in compiler
- First problem: identifying loops
  - Low level representation doesn't have loop statements!

Identifying loops

- First, we must identify *dominators*
  - Node a dominates node b if every possible execution path that gets to b *must* pass through a
- Many different algorithms to calculate dominators – we will not cover how this is calculated
  - Dataflow analysis?
- A *back edge* is an edge from b to a when a dominates b
- The target of a back edge is a *loop header*

Natural loops

- Will focus on *natural loops* – loops that arise in structured programs
- A node n is in a natural loop with header h
  - n must be dominated by h
  - There must be a path in the CFG from n to h through a back-edge to h
- What are the back edges in the example to the right? The loop headers? The natural loops?



next: loop invariant code motion