Global Optimizations

Overview

• Last lecture
  → Classical optimizations
  → Basic block construction
  → Basic block DAG construction

• This lecture
  → Dominators and loops
  → General code motion
  → Global common subexpression elimination

Scope of Optimization

• Peephole
  → Just 1-3 instructions
  → May be in different basic blocks

• Local
  → Within single basic block

• Global
  → Across basic blocks
  → Usually single procedure body

• Interprocedural
  → Across procedures

• Multiple versions of optimizations possible
  → Locally using DAG
  → Globally using available expressions dataflow analysis

Dominators and Loops

• Dominator
  → A node \( X \) dominates a node \( Y \) if and only if all paths from the control flow graph (CFG) entry node to \( Y \) pass through \( X \).
  → Note every node dominates itself.

• Loops
  → Let \((Y, X)\) be a CFG edge such that \( X \) dominates \( Y \).
  → Then all nodes on paths from \( X \) to \( Y \) are in the loop defined by \((Y, X)\).

• Intuitively
  → Loops can only be entered from one place
  → Examples

Dominators and Loops

What is the dominance relation?

\[
\begin{align*}
dom(1) &= \{1,2,3,4,5,6\} \\
dom(2) &= \{2,3,4,5,6\} \\
dom(3) &= \{3,4,5,6\} \\
dom(4) &= \{4,5\} \\
dom(5) &= \{5\} \\
dom(6) &= \{6\}
\end{align*}
\]

What are the loops?

\[
\begin{align*}
(5,3) &= \{3,4,5\} \\
(4,3) &= \{3,4\} \\
(6,2) &= \{2,3,4,5,6\}
\end{align*}
\]

Loops coalesced because same loop entry node (3)

Strength Reduction

\[
\begin{align*}
&i = 1 \\
i < n? \\
&exit\text{ }i := 1 \\
xtemp := x ** 1 \\
i < n? \\
&exit\text{ }s := x ** i \\
&... \\
i := i + 1 \\
xtemp := xtemp * x \\
i := i + 1
\end{align*}
\]
General Code Motion

```
n := 1; k := 0; m := 3; read x;
while n < 10 do
  if 2 + x ≥ 5 then k := 5;
  if 3 + k = 3 then m := m + 2;
  n := n + k + m;
endwhile;
```

Global Common Subexpression Elimination

```
z := a * b
r := 2 * z
```

Can be eliminated since a * b is available, i.e. calculated on all paths to this point.

```
w := a * b
```

Cannot be eliminated since a * b is not available on all paths reaching this point.

General Code Motion

```
n := 1; k := 0; m := 3; read x;
while n < 10 do
  if 2 + x ≥ 5 then k := 5;
  if 3 + k = 3 then m := m + 2;
  n := n + k + m;
endwhile;
```

Specialization of while loop depending on value of t1

```
n := 1; k := 0; m := 3; read x;
if 2 + x ≥ 5 then k := 5;
if 3 + k = 3 then m := m + 2;
else
  while n < 10 do
    m := m + 2;
    n := n + k + m;
  endwhile;
endif
```

Ensure a * b is assigned to the same variable t so it can be used for the assignment to u.
Forward Substitution

We can then forward substitute \( t \) for \( z \).

Dead Code Elimination

...and eliminate the assignment to \( z \) since it is now dead code.

What Else Can Be Done?

Partial Redundancy Elimination

We can compute \( a \times b \) on paths where it is not available.

Then eliminate the now fully redundant computation of \( a \times b \).