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**
  - Example: common subexpression elimination
    - Locally using DAG
    - Globally using available expressions dataflow analysis

**D 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
    - Loop
    - Not Loop
Dominators and Loops

- What is the dominance relation?
  - \( \text{dom}(1) = \{1, 2, 3, 4, 5, 6\} \)
  - \( \text{dom}(2) = \{2, 3, 4, 5, 6\} \)
  - \( \text{dom}(3) = \{3, 4, 5, 6\} \)
  - \( \text{dom}(4) = \{4, 5\} \)
  - \( \text{dom}(5) = \{5\} \)
  - \( \text{dom}(6) = \{6\} \)

- What are the loops?
  - \( (5, 3) = \{3, 4, 5\} \)
  - \( (4, 3) = \{3, 4\} \)
  - \( (6, 2) = \{2, 3, 4, 5, 6\} \)

Loops coalesced because of the same loop entry node (3)

Strength Reduction

```plaintext
i := 1

\text{exit} \quad i < n? \quad \text{exit}

s := x \cdot i \\
\ldots \\
i := i + 1

s := x \cdot i \\
\ldots \\
x := x \cdot i \\
i := i + 1
```
General Code Motion

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

Invariant within loop and therefore moveable.
Unaffected by definitions in loop and guarded by invariant condition
Moveable after we move statements 6 and 7.
Not moveable because may use def of m from statement 9 on previous iteration.

General Code Motion

1. n := 1;
2. k := 0;
3. m := 3;
4. read x;
5. while n < 10 do
6.    if 2 * x \geq 5 then k := 5;
7.    if 3 + k = 3 then m := m + 2;
8.    n := n + k + m;
9. endwhile;
General Code Motion

\[ \begin{align*}
n &:= 1; k := 0; m := 3; \text{read } x; \\
\text{while } n < 10 \text{ do} & \\
  & \quad \text{if } 2 \cdot x \geq 5 \text{ then } k := 5; \\
  & \quad \quad \text{if } 3 + k = 3 \text{ then } m := m + 2; \\
  & \quad n := n + k + m; \\
\text{endwhile}; \\
\end{align*} \]

\[ \begin{align*}
n &:= 1; k := 0; m := 3; \text{read } x; \\
  & \quad \text{if } 2 \cdot x \geq 5 \text{ then } k := 5; \\
  & \quad t1 := (3 + k = 3); \\
  & \quad \text{while } n < 10 \text{ do} \\
  & \quad \quad \text{if } t1 \text{ then } \\
  & \quad \quad \quad m := m + 2; \\
  & \quad \quad n := n + k + m; \\
  & \quad \text{endwhile; } \\
\end{align*} \]

Additional Optimization: Code Specialization

\[ \begin{align*}
n &:= 1; k := 0; m := 3; \text{read } x; \\
  & \quad \text{if } 2 \cdot x \geq 5 \text{ then } k := 5; \\
  & \quad t1 := (3 + k = 3); \\
  & \quad \text{if } t1 \text{ then} \\
  & \quad \quad \text{while } n < 10 \text{ do} \\
  & \quad \quad \quad m := m + 2; \\
  & \quad \quad n := n + k + m; \\
  & \quad \text{endif} \\
\end{align*} \]

Specialization of while loop depending on value of \( t1 \)
Global Common Subexpression Elimination

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

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

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?

\[
t := a \times b \\
r := 2 \times t \\
t := a \times b \\
q := t \\
u := t \\
z := u/2 \\
\]

Partial Redundancy Elimination

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

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