Compilers

Local Optimization
• The simplest form of optimization

• Optimize one basic block
  • No need to analyze the whole procedure body
• Some statements can be deleted
  \[
  x := x + 0 \\
  x := x * 1
  \]

• Some statements can be simplified
  \[
  x := x * 0 \Rightarrow x := 0 \\
  y := y ** 2 \Rightarrow y := y * y \\
  \begin{cases}
  x := x * 8 \\ x := x * 15
  \end{cases} \Rightarrow \begin{cases}
  x := x << 3 \\ t := x << 4; x := t - x
  \end{cases}
  \]

(On some machines \(<<\) is faster than \(*\); but not on all!)
Local Optimization

- Operations on constants can be computed at compile time
  - If there is a statement \( x := y \text{ op } z \)
  - And \( y \) and \( z \) are constants
  - Then \( y \text{ op } z \) can be computed at compile time

- Example: \( x := 2 + 2 \Rightarrow x := 4 \)
- Example: \( \text{if } 2 < 0 \text{ jump L can be deleted} \)
  \[ \text{false if } 2 > 0 \text{ jump L} \Rightarrow \text{jump L} \]
• Constant folding can be dangerous.

\[ a := \frac{1.5 + 3.7}{2} \]

\[ a := 5.2 \]

\[ a := 5.19 \]
• **Eliminate unreachable basic blocks:**
  – Code that is unreachable from the initial block
    • E.g., basic blocks that are not the target of any jump or “fall through” from a conditional

• **Removing unreachable code makes the program smaller**
  – And sometimes also faster
    • Due to memory cache effects
    • Increased spatial locality
Local Optimization

• Why would unreachable basic blocks occur?

```
#define DEBUG 0

if (DEBUG) then

==>

if (0) then
```

libraries

result of optimizations
• Some optimizations are simplified if each register occurs only once on the left-hand side of an assignment

• Rewrite intermediate code in **single assignment** form

\[
\begin{aligned}
x &:= z + y \\
a &:= x \\
x &:= 2 \cdot x
\end{aligned} \quad \Rightarrow \quad \begin{aligned}
b &:= z + y \\
a &:= b \\
x &:= 2 \cdot b
\end{aligned}
\]

(b is a fresh register)

– More complicated in general, due to loops
Local Optimization

• If
  – Basic block is in single assignment form
  – A definition \( x := \) is the first use of \( x \) in a block

• Then
  – When two assignments have the same rhs, they compute the same value

• Example:

\[
\begin{align*}
  x &:= y + z \\
  \cdots \quad \Rightarrow \quad \cdots \\
  w &:= y + z
\end{align*}
\]

\[
\begin{align*}
  x &:= y + z \\
  \cdots \quad \Rightarrow \quad \cdots \\
  w &:= x
\end{align*}
\]

(the values of \( x \), \( y \), and \( z \) do not change in the \( \cdots \) code)
Local Optimization

• If \( w := x \) appears in a block, replace subsequent uses of \( w \) with uses of \( x \)
  – Assumes single assignment form

• Example:

\[
\begin{align*}
  b & := z + y \\
  a & := b \\
  x & := 2 * a
\end{align*}
\]

\[
\begin{align*}
  b & := z + y \\
  a & := \text{copy} \\
  x & := 2 * \text{propagation}
\end{align*}
\]

• Only useful for enabling other optimizations
  – Constant folding
  – Dead code elimination
• Example:

\[
\begin{align*}
a & := 5 \\
x & := 2 \times 5 \\
y & := x + 6 \\
t & := x \times y
\end{align*}
\]

\[
\begin{align*}
a & := 5 \\
x & := 10 \\
y & := 16 \\
t & := x \ll 4
\end{align*}
\]
If \( w := \text{rhs} \) appears in a basic block and \( w \) does not appear anywhere else in the program

Then

the statement \( w := \text{rhs} \) is dead and can be eliminated

– Dead = does not contribute to the program’s result

Example: (\( a \) is not used anywhere else)

\[
\begin{align*}
    x := z + y & \quad b := z + y & \quad b := z + y \\
    a := x & \quad \Rightarrow \quad a := b & \quad \Rightarrow \quad x := 2 \times b \\
    x := 2 \times a & \quad x := 2 \times b
\end{align*}
\]
• Each local optimization does little by itself

• Typically optimizations interact
  – Performing one optimization enables another

• Optimizing compilers repeat optimizations until no improvement is possible
  – The optimizer can also be stopped at any point to limit compilation time
• Initial code:

```plaintext
a := x ** 2
b := 3
c := x
d := c * c
e := b * 2
f := a + d
g := e * f
```
• Algebraic optimization:

\[ a := x^{**} 2 \]
\[ b := 3 \]
\[ c := x \]
\[ d := c \ast c \]
\[ e := b \ast 2 \]
\[ f := a + d \]
\[ g := e \ast f \]
• Algebraic optimization:

\[
\begin{align*}
a &:= x \times x \\
b &= 3 \\
c &= x \\
d &= c \times c \\
e &= b \ll 1 \\
f &= a + d \\
g &= e \times f
\end{align*}
\]
Copy propagation:

\[
\begin{align*}
a & := x \times x \\
b & := 3 \\
c & := x \\
d & := c \times c \\
e & := b \ll 1 \\
f & := a + d \\
g & := e \times f
\end{align*}
\]
Local Optimization

• **Copy propagation:**

```
a := x * x
b := 3
c := x
d := x * x
e := 3 << 1
f := a + d
g := e * f
```
• Constant folding:

\[
\begin{align*}
a & := x \times x \\
b & := 3 \\
c & := x \\
d & := x \times x \\
e & := 3 \ll 1 \\
f & := a + d \\
g & := e \times f
\end{align*}
\]
Local Optimization

- **Constant folding:**

\[
\begin{align*}
a & := x \times x \\
b & := 3 \\
c & := x \\
d & := x \times x \\
e & := 6 \\
f & := a + d \\
g & := e \times f
\end{align*}
\]
Local Optimization

- **Common subexpression elimination:**

  ```
  a := x * x
  b := 3
  c := x
  d := x * x
  e := 6
  f := a + d
  g := e * f
  ```
• Common subexpression elimination:

```
\[ a := x \times x \]
\[ b := 3 \]
\[ c := x \]
\[ d := a \]
\[ e := 6 \]
\[ f := a + d \]
\[ g := e \times f \]
```
Copy propagation:

\[
\begin{align*}
a &:= x \times x \\
b &:= 3 \\
c &:= x \\
d &:= a \\
e &:= 6 \\
f &:= a + d \\
g &:= e \times f
\end{align*}
\]
• Copy propagation:

\[
\begin{align*}
a &:= x \times x \\
b &:= 3 \\
c &:= x \\
d &:= a \\
e &:= 6 \\
f &:= a + a \\
g &:= 6 \times f
\end{align*}
\]
• Dead code elimination:

\[
\begin{align*}
a &:= x \times x \\
b &:= 3 \\
c &:= x \\
d &:= a \\
e &:= 6 \\
f &:= a + a \\
g &:= 6 \times f
\end{align*}
\]
• Dead code elimination:
  
  \[
a := x \times x
\]

  \[
f := a + a
\]

  \[
g := 6 \times f
\]

  \[
g := 12 \times a
\]

• This is the final form
Which of the following are valid local optimizations for the given basic block? Assume that only $g$ and $x$ are referenced outside of this basic block.

- Copy propagation: Line 4 becomes $d := a \times b$.
- Common subexpression elimination: Line 5 becomes $e := d$.
- Dead code elimination: Line 3 is removed.
- After many rounds of valid optimizations, the entire block can be reduced to $g := 5$. 

**Local Optimization**

```
1  a := 1
2  b := 3
3  c := a + x
4  d := a \times 3
5  e := b \times 3
6  f := a + b
7  g := e - f
```