Local Optimization

Lecture 24

Dr. Sean Peisert – ECS 142 – Spring 2009
Status

• Project 3 due today
• Project 4 due June 5, 11:55pm
• Office Hours today at 11am
• No Class (Memorial Day) on Monday
• Read Sec. 8.1 through 8.8 by Wednesday, May 27
Local Optimizations

• The simplest form of optimizations
• No need to analyze the whole procedure body
  • Just the basic block in question
• Example: algebraic simplification
Algebraic Simplification

- Some statements can be deleted
  - $x := x + 0$
  - $x := x \times 1$

- Some statements can be simplified
  - $x := x \times 0 \Rightarrow x := 0$
  - $y := y \times 2 \Rightarrow y := y \times y$
  - $x := x \times 8 \Rightarrow x := x \ll 3$
  - $x := x \times 15 \Rightarrow t := x \ll 4; x := t - x$
  - (on some machines $\ll$ is faster than $\times$; but not on all!)
Constant Folding

- Operations on constants can be computed at compile time
- In general, 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: if \( 2 < 0 \) jump L can be deleted
- When might constant folding be dangerous?

Friday, May 22, 2009
Flow of Control Optimizations

• Eliminating unreachable code:
  • Code that is unreachable in the control-flow graph
  • Basic blocks that are not the target of any jump or “fall through” from a conditional
  • Such basic blocks can be eliminated
• Why would such basic blocks occur?
• Removing unreachable code makes the program smaller
  • And sometimes also faster
    • Due to memory cache effects (increased spatial locality)
Single Assignment Form

- Some optimizations are simplified if each register occurs only once on the left-hand side of an assignment.
- Intermediate code can be rewritten to be in single assignment form:
  \[
  x := z + y \quad \Rightarrow \quad a := x \\
  b := z + y \quad \Rightarrow \quad a := b \\
  x := 2 \times x \quad \Rightarrow \quad x := 2 \times b
  \]
  (b is a fresh register)
- More complicated in general, due to loops.
Common Subexpression Elimination

- Assume
  - Basic block is in single assignment form
  - A definition $x :=$ is the first use of $x$ in a block
- All assignments with same rhs compute the same value
- Example:
  \[
  x := y + z \quad \Rightarrow \quad x := y + z \\
  \ldots \quad \Rightarrow \quad \ldots \\
  w := y + z \quad \Rightarrow \quad w := x \\
  \]
- (the values of $x$, $y$, and $z$ do not change in the...code)
Copy Propagation

• If \( w := x \) appears in a block, all subsequent uses of \( w \) can be replaced with uses of \( x \)

• Example:
  \[
  b := z + y \\
  a := b \\
  \Rightarrow \\
  x := 2 * a \\
  x := 2 * b
  \]

• This does not make the program smaller or faster but might enable other optimizations
  • – Constant folding
  • – Dead code elimination
Copy Propagation and Constant Folding

Example:

\[ a := 5 \]
\[ x := 2 \times a \quad \Rightarrow \quad x := 10 \]
\[ y := x + 6 \quad \Rightarrow \quad y := 16 \]
\[ t := x \times y \quad \Rightarrow \quad t := x \ll 4 \]
Copy Propagation and Dead Code Elimination

- If
  - \( w := \text{rhs} \) appears in a basic block
  - \( w \) does not appear anywhere else in the program
- Then
  - the statement \( w := \text{rhs} \) is dead and can be eliminated
  - \( \text{Dead} = \) does not contribute to the program’s result
- Example: (\( a \) is not used anywhere else)

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

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

• Each local optimization does very little by itself
• Typically optimizations interact
  • Performing one optimizations enables other opt.
• Typical optimizing compilers repeatedly perform optimizations until no improvement is possible
• The optimizer can also be stopped at any time to limit the compilation time
Peephole Optimizations on Assembly Code

• The optimizations presented before work on intermediate code
• They are target independent
• But they can be applied on assembly language also
• Peephole optimization is an effective technique for improving assembly code
• The “peephole” is a short sequence of (usually contiguous) instructions
• The optimizer replaces the sequence with another equivalent one (but faster)
Peephole Optimizations

• Write peephole optimizations as replacement rules
  \[ i_1, \ldots, i_n \rightarrow j_1, \ldots, j_m \]
• where the rhs is the improved version of the lhs
• Example:
  \[ \text{move } a \ b, \text{move } b \ a \rightarrow \text{move } a \ b \]
    • Works if move $b \ a$ is not the target of a jump
  • Another example
    \[ \text{addiu } a \ a \ i, \text{addiu } a \ a \ j \rightarrow \text{addiu } a \ a \ i+j \]
Peephole Optimizations

- Many (but not all) of the basic block optimizations can be cast as peephole optimizations
- Example: `addiu $a $b 0 → move $a $b`
- Example: `move $a $a →`  
- These two together eliminate `addiu $a $a 0`
- Just like for local optimizations, peephole optimizations need to be applied repeatedly to get maximum effect
Local Optimizations. Notes.

- Intermediate code is helpful for many optimizations
- Many simple optimizations can still be applied on assembly language
- “Program optimization” is grossly misnamed
  - Code produced by “optimizers” is not optimal in any reasonable sense
  - “Program improvement” is a more appropriate term
- Next time: global optimizations
Global Optimization
Local Optimization

- Recall the simple basic-block optimizations
  - Constant propagation
  - Dead code elimination

\[
\begin{align*}
X & := 3 \\
Y & := Z \ast W \\
Q & := X + Y
\end{align*}
\]

\[
\begin{align*}
X & := 3 \\
Y & := Z \ast W \\
Q & := 3 + Y
\end{align*}
\]
Global Optimization
Diagrams
Finish Project 3!
Start Project 4!