CS 536 Announcements for Wednesday, April 24, 2024

Course evaluation – log into heliocampusac.wisc.edu using your NetID

Last Time
• optimization overview
• peephole optimization
• loop optimizations

Today
• wrap up optimization
• copy propagation

Optimization Review

Goal: Produce "better" code that does the "same thing" as the original code.
• better = faster code, fewer instructions
• same thing = determined by observable behavior of code

When?
• before code generation (i.e., on intermediate representation)
• after code generation (i.e., on generated machine code)

Important considerations
• performance/profitability – want to be sure optimization is "worth it"
• safety – orginal source code, non-optimized target code, and optimized target code all do the "same thing" / have the same "meaning"

Look at optimizations that
• are sound transformations sound = all results that are output are valid
• recognize a behavior in a program & replace it with a "better" version
Copy propagation

**copy statement**
- **definition** of $x$: $x = y$.
- **use** of $y$:

**Idea:** Suppose we are at **use** $U$ of $x$ and a **definition** $D$ of $x$ (of the form $x = y$) reaches $U$
- If
  1) no other definition of $x$ reaches $U$ and
  2) $y$ does not change between $D$ and $U$
- then we can replace the use of $x$ at $U$ with $y$

**Example**

- $x = 3$.
- $y = 5$.
- $p = 3$.
- if $w*x > 9$ [\[ $x = 4$.  \\
   z = x + w*y$. ]
- else [\[ $z = 2*y + x$. ]
- $q = 5*p$.
- $s = z + x$.
- $t = s + 5$.
How is this an optimization?

- can create **useless code** (which can then be removed)

  \[
  \text{if all uses of } x \text{ reach by } D \text{ are replaced,}
  \]
  \[
  \text{then definition } D \text{ can be removed } (e.g., y = 5)
  \]

- can create improved code

  \[
  t = s + y.
  \]
  \[
  \text{RHS requires (at a minimum) 2 loads & 1 add}
  \]
  \[
  t = s + 5.
  \]
  \[
  \text{RHS requires only 1 load & 1 add}
  \]
  \[
  \text{(can use immediate value in add instr)}
  \]

- **constant folding**

  \[
  z = 2 \times 5 + 3. \rightarrow z = 10 + 3 \rightarrow z = 13
  \]
  \[
  \text{now can copy propagate this def of } z
  \]

- if done before other optimizations, can improve results
Copy propagation (cont.)

Recall: Suppose we are at use \( U \) of \( x \) and a definition \( D \) of \( x \) (of the form \( x = y \)) reaches \( U \)

- If
  1) no other definition of \( x \) reaches \( U \) and
  2) \( y \) does not change between \( D \) and \( U \)
- then we can replace the use of \( x \) at \( U \) with \( y \)

So, to do copy propagation, we must make sure two properties hold:

**Property 1)** No other definition of \( x \) reaches \( U \)

**Property 2)** \( y \) does not change between \( D \) and \( U \)

How?

**Property 1)** No other definition of \( x \) reaches \( U \)

- How? Do a **reaching-definitions** analysis
  - one way: data flow analysis
  - another way: create control flow graph (CFG)

- do "backwards" search starting at \( U \)
- stop exploring a branch of a search when we find a \( \text{def} \) of \( x \)
  (but continue overall search)
\[ x = 3. \]
\[ y = w. \]
\[ p = x. \]

if \( w \times x > 9 \) [  
\[ x = 4. \]

while \( x < 10 \) [  
\[ z = x + w \times y. \]
\[ x = x + 1. \]
]  
else [  
\[ z = 2 \times y + x. \]
]  

\[ q = 5 \times p. \]
\[ s = z + x. \]
\[ t = s + y. \]
Copy Propagation (cont.)

Property 2) \( y \) does not change between D and U (of x)

- If \( y \) is a constant, then this is trivially true.

- If on any path through the CFG from D to U there is a definition of \( y \), then

\[
\text{\( y \) might change}
\]

- If \( y \) and \( z \) are aliases and there is a definition of \( z \) between D and U, then

\[
\text{\( y \) might change} \quad \text{\( z \) refer to same spot in memory}
\]

\[
X = y \downarrow \\
\text{\( z \) \( \downarrow \) code to make \( y \) \& \( z \) aliases} \\
Z = 5 \downarrow \\
W = x + y \downarrow \\
\text{\( \downarrow \) can't replace \( x \) with \( y \)}
\]

In C/C++

\[
X = y \downarrow \\
\text{\( int \ *z = \&y \downarrow \)} \\
\text{\( *z = \text{\( 5 \downarrow \)}} \\
w = x + y \downarrow \\
\text{\( *z \& y \) are same place in memory}
\]
Example (cont.)

\[
x = 3.
\]

\[
y = w.
\]

\[
p = x^3.
\]

\[
\text{if } w * x > 9 \text{ [}
\]

\[
x = 4.
\]

\[
\text{while } x < 10 \text{ [}
\]

\[
z = x + w * x^3.
\]

\[
x = x + 1.
\]

\]

\[
\text{else [}
\]

\[
z = 2 * w^3 + x^3.
\]

\]

\[
q = 5 * p. \text{— on a 2nd pass } p(1) \text{ reaches}
\]

\[
\text{and can } \text{copy prop} \text{(the constant 3)} \text{ to get: } q = 5 * 3.
\]

\[
s = z + x.
\]

\[
t = s + y^w.
\]

Optimization Wrap-up

Back end

IR optimizer

\[
\text{copy propagation}
\]

code generator

MC optimizer

\[
\text{peephole optimizations}
\]