Where are we?

- Analysis
  - Control Flow/Predicate
    - Treat basic blocks as a black box
    - Only look at branches
  - Dataflow
    - Look inside basic blocks
    - What is computed where?

Static Single Assignment (SSA) Form

- Each assignment to a variable is given a unique name
- All of the uses reached by that assignment are renamed
  - Trivial for straight line code
    - $x = -1$
    - $y = x$
    - $x = 5$
    - $z = x$
- More complex with control flow – Must use Phi nodes
  - if ($...$)
    - $x = -1$
  - else
    - $x = 5$
  - $y = x$

SSA Overview (continued)

- What about loops – no problem
  - $i = 0$
  - do {
    - $i = i + 1$
  } while ($i < 50$)

- Advantages of SSA
  - Explicit DU chains – Trivial to figure out what defs reach a use
    - Each use has exactly 1 definition!!!
  - Explicit merging of values
  - Makes optimizations easier

- Disadvantages
  - When transform the code, must either recompute (slow) or incrementally update (tedious)
Phi Nodes (aka Phi Functions)

- Special kind of copy that selects one of its inputs
- Choice of input is governed by the CFG edge along which control flow reached the Phi node

\[
x_0 = x_1 = \quad x_2 = \text{Phi}(x_0, x_1)
\]

- Phi nodes are required when 2 non-null paths X→Z and Y→Z converge at node Z, and nodes X and Y contain assignments to V

SSA Construction

- High-level algorithm
  1. Insert Phi nodes
  2. Rename variables
  3. Profit 😊
- A dumb algorithm
  - Insert Phi functions at every join for every variable
  - Solve reaching definitions
  - Rename each use to the def that reaches it (will be unique)
- Problems with the dumb algorithm
  - Too many Phi functions (precision)
  - Too many Phi functions (space)
  - Too many Phi functions (time)

Need Better Phi Node Insertion Algorithm

- A definition at n forces a Phi node at m iff
  - n not in DOM(m), but n in DOM(p) for some predecessors p of m

Dominator Tree

- First BB is the root node, each node dominates all of its descendants

<table>
<thead>
<tr>
<th>BB</th>
<th>DOM</th>
<th>BB</th>
<th>DOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.1,3,4</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>5</td>
<td>0.1,3,5</td>
</tr>
<tr>
<td>2</td>
<td>0.1,2</td>
<td>6</td>
<td>0.1,3,6</td>
</tr>
<tr>
<td>3</td>
<td>0.1,3</td>
<td>7</td>
<td>0.1,7</td>
</tr>
</tbody>
</table>

def in BB4 forces Phi in BB6
def in BB6 forces Phi in BB7
def in BB7 forces Phi in BB1

Phi is placed in the block that is just outside the dominated region of the definition BB

Dominance frontier
The dominance frontier of node X is the set of nodes Y such that
* X dominates a predecessor of Y, but
* X does not strictly dominate Y
For each join point X in the CFG
For each predecessor of X in the CFG
Run up to the IDOM(X) in the dominator tree,
adding X to DF(N) for each N between X and IDOM(X)

Phi Node Insertion Algorithm

- Compute dominance frontiers
- Find global names (aka virtual registers)
  - Global if name live on entry to some block
  - For each name, build a list of blocks that define it
- Insert Phi nodes
  - For each global name n
    - For each BB b in which n is defined
      - For each BB d in b’s dominance frontier
        - Insert a Phi node for n in d
        - Add d to n’s list of defining BBs

Example

For each BB, calculate the Phi nodes needed for each global name.

BB  DF
0  -
1  -
2  7
3  7
4  6
5  6
6  7
7  1
Class Problem

Insert the Phi nodes

COS 598C - Advanced Compilers 12 Prof. David August

Renaming Variables – Pseudo Code

- Main function
  - For each global name i
    - counter[i] = 0
    - stack[i] = NULL
  - call Rename(BB0)

- Rename(b)
  - For each Phi node in b, x = Phi()
    - Rename x as NewName(x)
  - For each operation, x = y op z, in b
    - y, z = top(stack[y]), top(stack[z])
    - x = NewName(x)
  - For each successor of b in the CFG
    - Set appropriate Phi parameters
  - For each successor s of b in dom tree
    - Rename(s)
  - For each operation, x = y op z, in b
    - pop(stack[x])

- NewName(n)
  - i = counter[n]
  - counter[n]++
  - push ni onto stack[n]
  - return ni

SSA Step 2 – Renaming Variables

- Use an array of stacks, one stack per global variable (VR)
- Algorithm sketch
  - For each BB b in a preorder traversal of the dominator tree
    - Generate unique names for each Phi node
    - Rewrite each operation in the BB
      - Uses of global name: current name from stack
      -_defs of global name: create and push new name
    - Fill in Phi node parameters of successor blocks
    - Recurse on b’s children in the dominator tree
    - <on exit from b> pop names generated in b from stacks

Renaming – Example (Initial State)

var: a b c d i
ctr: 0 0 0 0 0
stk: a0 b0 c0 d0 i0

COS 598C - Advanced Compilers 14 Prof. David August
Renaming – Example (After BB0)

\[
\begin{align*}
\text{BB0} & \quad \text{BB1} \\
\text{BB2} & \quad \text{BB3} \\
\text{BB4} & \quad \text{BB5} \\
\text{BB6} & \quad \text{BB7}
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_0, a) \\
b & = \Phi(b_0, b) \\
c & = \Phi(c_0, c) \\
d & = \Phi(d_0, d) \\
i & = \Phi(i_0, i)
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_2, a) \\
b & = \Phi(b_2, b) \\
c & = \Phi(c_3, c) \\
d & = \Phi(d_2, d)
\end{align*}
\]

Renaming – Example (After BB1)

\[
\begin{align*}
\text{BB0} & \quad \text{BB1} \\
\text{BB2} & \quad \text{BB3} \\
\text{BB4} & \quad \text{BB5} \\
\text{BB6} & \quad \text{BB7}
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_0, a) \\
b & = \Phi(b_0, b) \\
c & = \Phi(c_0, c) \\
d & = \Phi(d_0, d) \\
i & = \Phi(i_0, i)
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_2, a) \\
b & = \Phi(b_2, b) \\
c & = \Phi(c_3, c) \\
d & = \Phi(d_2, d)
\end{align*}
\]

Renaming – Example (After BB2)

\[
\begin{align*}
\text{BB0} & \quad \text{BB1} \\
\text{BB2} & \quad \text{BB3} \\
\text{BB4} & \quad \text{BB5} \\
\text{BB6} & \quad \text{BB7}
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_0, a) \\
b & = \Phi(b_0, b) \\
c & = \Phi(c_0, c) \\
d & = \Phi(d_0, d) \\
i & = \Phi(i_0, i)
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_2, a) \\
b & = \Phi(b_2, b) \\
c & = \Phi(c_3, c) \\
d & = \Phi(d_2, d)
\end{align*}
\]

Renaming – Example (Before BB3)

\[
\begin{align*}
\text{BB0} & \quad \text{BB1} \\
\text{BB2} & \quad \text{BB3} \\
\text{BB4} & \quad \text{BB5} \\
\text{BB6} & \quad \text{BB7}
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_0, a) \\
b & = \Phi(b_0, b) \\
c & = \Phi(c_0, c) \\
d & = \Phi(d_0, d) \\
i & = \Phi(i_0, i)
\end{align*}
\]

\[
\begin{align*}
\alpha & = \Phi(a_2, a) \\
b & = \Phi(b_2, b) \\
c & = \Phi(c_3, c) \\
d & = \Phi(d_2, d)
\end{align*}
\]
Renaming – Example (After BB3)

\[ a_2 = c_2 = b_2 = c_3 = d_2 = a_3 = d_3 = \]
\[ c_1 = \Phi(a_0, a) \]
\[ b_1 = \Phi(b_0, b) \]
\[ c_1 = \Phi(c_0, c) \]
\[ d_1 = \Phi(d_0, d) \]
\[ i_1 = \Phi(i_0, i) \]

```
var: a b c d i
ctr: 4 3 4 2
stk: a0 b0 c0 d0 i0
a1 b1 c1 d1 i1
a2 c2 d3
a3
```

Renaming – Example (After BB4)

\[ a_2 = c_2 = b_2 = c_3 = d_2 = a_3 = d_3 = \]
\[ c_4 = d_4 = b_3 = i = \]
\[ a_1 = \Phi(a_0, a) \]
\[ b_1 = \Phi(b_0, b) \]
\[ c_1 = \Phi(c_0, c) \]
\[ d_1 = \Phi(d_0, d) \]
\[ i_1 = \Phi(i_0, i) \]

```
var: a b c d i
ctr: 4 3 4 5 2
stk: a0 b0 c0 d0 i0
a1 b1 c1 d1 i1
a2 c2 d3
a3 d4
```

Renaming – Example (After BB5)

\[ a_2 = c_2 = b_2 = c_3 = d_2 = a_3 = d_3 = c_4 = d_4 = \]
\[ b_3 = \]
\[ a_1 = \Phi(a_0, a) \]
\[ b_1 = \Phi(b_0, b) \]
\[ c_1 = \Phi(c_0, c) \]
\[ d_1 = \Phi(d_0, d) \]
\[ i_1 = \Phi(i_0, i) \]

```
var: a b c d i
ctr: 4 3 5 5 2
stk: a0 b0 c0 d0 i0
a1 b1 c1 d1 i1
a2 c2 d3
a3 c4
```

Renaming – Example (After BB6)

\[ a_2 = c_2 = b_2 = c_3 = d_2 = a_3 = d_3 = c_4 = d_4 = \]
\[ b_3 = \]
\[ a_1 = \Phi(a_0, a) \]
\[ b_1 = \Phi(b_0, b) \]
\[ c_1 = \Phi(c_0, c) \]
\[ d_1 = \Phi(d_0, d) \]
\[ i_1 = \Phi(i_0, i) \]

```
var: a b c d i
ctr: 4 4 6 6 2
stk: a0 b0 c0 d0 i0
a1 b1 c1 d1 i1
a2 b3 c2 d3
a3 c5 d5
```
Renaming – Example (After BB7) – Dats it!

- **BB0**: 
  - \( a_0 = b_0 = c_0 = d_0 = i_0 = \) PHI(0,0,0)

- **BB1**: 
  - \( a_1 = \) PHI(a0, a4)
  - \( b_1 = \) PHI(b0, b4)
  - \( c_1 = \) PHI(c0, c6)
  - \( d_1 = \) PHI(d0, d6)
  - \( i_1 = \) PHI(i0, i2)

- **BB2**: 
  - \( a_2 = b_2 = c_2 = d_2 = \)

- **BB3**: 
  - \( a_3 = \) PHI(a0, a4)
  - \( b_3 = \) PHI(b0, b4)
  - \( c_3 = \) PHI(c0, c6)
  - \( d_3 = \) PHI(d0, d6)

- **BB4**: 
  - \( a_4 = \) PHI(a2, a3)
  - \( b_4 = \) PHI(b2, b3)
  - \( c_4 = \) PHI(c3, c5)
  - \( d_4 = \) PHI(d2, d5)

- **BB5**: 
  - \( c_5 = \) PHI(c2, c4)
  - \( d_5 = \) PHI(d4, d3)

- **BB6**: 
  - \( b_3 = \)

- **BB7**: 
  - \( c_2 = \)

---

Class Problem

Rename the variables so this code is in SSA form:

- **BB0**: 
  - \( a = \) PHI(a0, a4)
  - \( b = \) PHI(b0, b4)
  - \( c = \) PHI(c0, c6)
  - \( d = \) PHI(d0, d6)
  - \( i = \) PHI(i0, i2)

- **BB1**: 
  - \( a = \) PHI(a0, a4)
  - \( b = \) PHI(b0, b4)
  - \( c = \) PHI(c0, c6)
  - \( d = \) PHI(d0, d6)

- **BB2**: 
  - \( c = \) PHI(c2, c4)
  - \( d = \) PHI(d4, d3)

- **BB3**: 
  - \( a = \) PHI(a2, a3)
  - \( b = \) PHI(b2, b3)
  - \( c = \) PHI(c3, c5)
  - \( d = \) PHI(d2, d5)

- **BB4**: 
  - \( b = \)

- **BB5**: 
  - \( c = \)