More CFG
Static Single Assignment
Announcements

- HW 1 due next Monday (9/17)
- Lab 2 will be released tonight (due 9/24)
- Instructor OH cancelled this afternoon
Outline

- More on control flow analysis
  - Dominance frontier

- Dataflow analysis – static single assignment (SSA)
  - SSA Definition
  - PHI node (Φ-node) placement
  - Code optimizations with SSA
Revisiting Control Flow Analysis

- What becomes a leader statement of a basic block? (three cases)

- Does a node strictly (or properly) dominate itself? \(\textbf{No}\)

- Does a predecessor of a node \(B\) always dominate \(B\)? \(\textbf{No}\)

- Suppose \(A\) that dominates all of \(B\)’s predecessors. Does \(A\) always dominate \(B\)? \(\textbf{Yes}\)
Review: Finding Loops

- Loop identification algorithm
  - Find an edge $B \rightarrow H$ where $H$ dominates $B$;
    This edge is called a **back-edge**
  - Find all nodes that (1) are dominated by $H$ and (2) can reach $B$ through nodes dominated by $H$; add them to the loop
    - $H$ and $B$ are naturally included
Finding Loops (1)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)
Finding Loops (1)

Find all back edges in this graph and the natural loop associated with each back edge

$(9,1)$ Entire graph
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7)
Intuition of Dominance Relation

Imagine a source of light at the entry node, and that the edges are optical fibers.

To find which nodes are dominated by a given node, place an opaque barrier at that node and observe which nodes become dark.
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)  Entire graph
(10,7)
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)  Entire graph
(10,7) {7,8,10}
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7) \{7,8,10\}
(7,4)
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)   Entire graph
(10,7)  \{7,8,10\}
(7,4)
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7) \{7,8,10\}
(7,4) \{4,5,6,7,8,10\}
Static Single Assignment

- Static single assignment (SSA) form is a restricted IR where
  - Each variable definition has a unique name
  - Each variable use refers to a single definition

- SSA simplifies data flow analysis & many compiler optimizations
  - Eliminates artificial dependences (on scalars)
    - Write-after-write
    - Write-after-read
SSA within a Basic Block

- Assign each variable definition a unique name
- Update the uses accordingly

```
x = read()
x = x * 5
x = x + 1
y = x * 9
```

```
x_0 = read()
x_1 = x_0 * 5
x_2 = x_1 + 1
y = x_2 * 9
```

Corresponding data flow graph
Consider a situation where two control-flow paths merge:
- e.g., due to an if-then-else statement or a loop

```
x = read()
if (x > 0)
y = 5
else
 y = 10
x = y
```

Should this be $y_0$ or $y_1$?
Introducing $\phi$-Node

- Inserts special join functions (called $\phi$-nodes or PHI nodes) at points where different control flow paths converge

Note: $\phi$ is not an executable function!

To generate executable code from this form, appropriate copy statements need to be generated in the predecessors (in other words, reversing the SSA process for code generation)
SSA in a Loop

- Insert $\phi$-nodes in the loop header block

```
x = 0
i = 1
while (i < 10) {
    x = x + i
    i = i + 1
}
```
ϕ-Node Placement

When and where to add ϕ-nodes?

- If two control paths A → C and B → C converge at a node C, and both A and B contain assignments to variable X, then ϕ-node for X must be placed at C
  
  • We often call C a join node or convergence point
  
  • Can be generalized to more than two converging control paths

Objective: Minimize the number of ϕ-nodes

- Need to compute dominance frontier sets
Dominance Frontier

➢ A basic block $F$ is in the dominance frontier set (DF) of basic block $B$ if and only if
  - $B$ does NOT strictly dominate $F$
  - $B$ dominates some predecessor(s) of $F$

If above two conditions hold, $F \in \text{DF}(B)$

➢ Intuitively, the basic blocks in the dominance frontier set of $B$ are *almost* strictly dominated by $B$

➢ Useful for efficiently computing the SSA form
Dominance Frontier and SSA

- **B7 is in the dominance frontier set of B3**
  - B3 does not dominate B7, but dominates one of its predecessors (i.e., B6)

- **For each variable definition in B3, a \( \phi \) node is needed in B7**
  - B7 is the destination of some edge(s) leaving an area dominated by B3
Example: Dominance Frontiers

- $F$ is in the dominance frontier set (DF) of $B$ iff
  - $B$ does NOT strictly dominate $F$
  - $B$ dominates some predecessor(s) of $F$

If above two conditions hold, $F \in \text{DF}(B)$
Iterative $\phi$-Node Insertion

<table>
<thead>
<tr>
<th>B</th>
<th>DF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

- $a$ is defined in 0, 3  
  $\implies$ need $\phi$ in 7,  
  then $a$ defined in 7

- $b$ is defined in 0, 2, 6  
  $\implies$ need $\phi$ in 7  
  then $b$ defined in 7

- $c$ is defined in 0, 2, 5  
  $\implies$ need $\phi$ in 6, 7  
  then $c$ defined in 7
SSA Applications

- SSA form simplifies data flow analysis and many code transformations
  - Primarily due to explicit & simplified (sparse) def-use chains

- Here we show two simple examples
  - Dead code elimination
  - Loop induction variable detection
Dead Code in CDFG

- Dead code is either
  - Unreachable code
  - Definitions never used

- Dead statements?
Dead Code Elimination with SSA

Iteratively remove unused definitions:
remove y1, z2 (and B4) → then remove z1
Loop Induction Variables

- An induction variable is a variable that
  
  - Gets increased or decreased by a fixed amount (loop invariant) on every iteration of a loop (basic induction variable)
    
    • \( i = i + c \)

  - or is an affine function of another induction variable (mutual induction variable)
    
    • \( j = a \times i + b \)
Find basic loop induction variable(s)

1. Inspect back edges in the loop

2. Each back edge points to a $\phi$ node in the loop header, which may indicate a basic induction variable

3. $\phi$ is a function of an initialized variable and a definition in the form of "$i + c$" (i.e., increment operation)
Summary

- Importance of compilers
  - Essential component of SoC software development flow
  - Essential component of high-level synthesis

- A good intermediate representation (IR) enables efficient and effective analysis and optimization
  - Dominance relation helps effective CFG analysis
  - SSA form facilitates efficient IR-level optimization
Acknowledgements

- These slides contain/adapt materials developed by
  - Prof. Scott Mahlke (UMich)