

Using Static Single Assignment

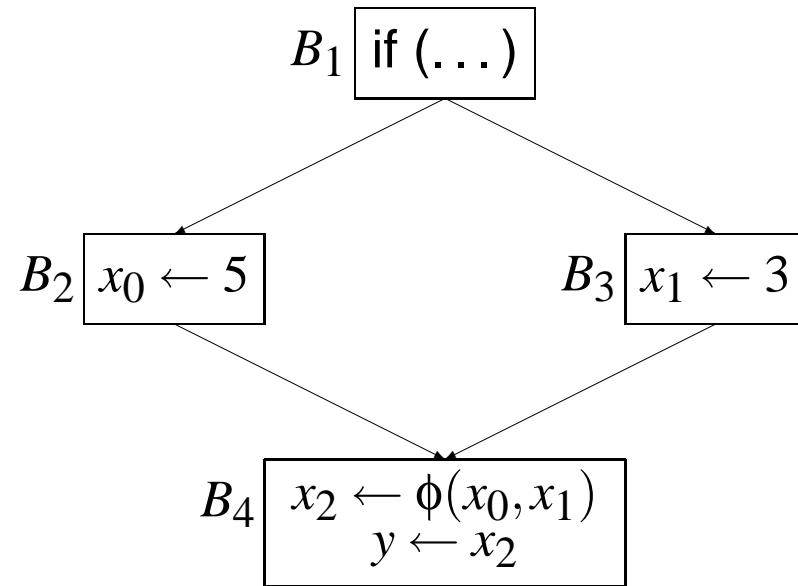
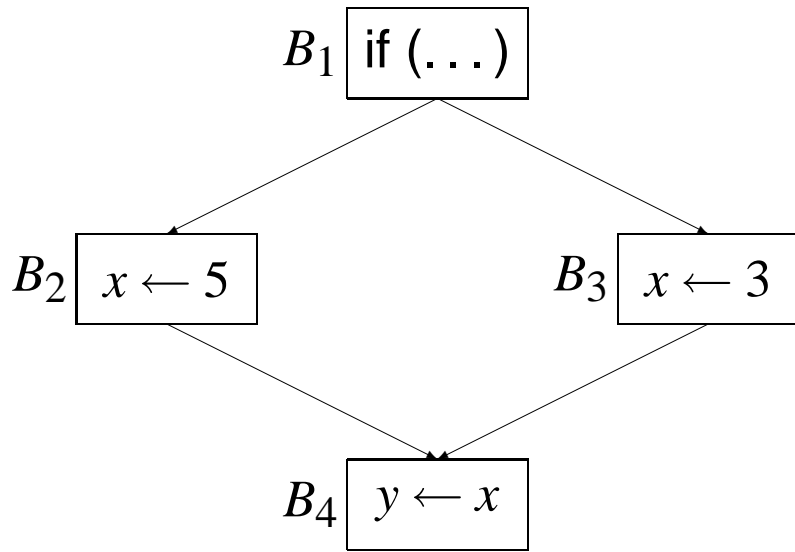
Last Time

- Basic definition, and why it is useful
- How to build it

Today

- Loop Optimizations
 - Induction variables (standard vs. SSA)
 - Loop Invariant Code Motion (SSA based)

SSA form



Loop Optimization

Loops are important, they execute often

- typically, some regular access pattern

regularity \Rightarrow opportunity for improvement

repetition \Rightarrow savings are multiplied

- *assumption*: loop bodies execute 10^{depth} times

Classical Loop Optimizations

- Loop Invariant Code Motion
- Induction Variable Recognition
- Strength Reduction
- Linear Test Replacement
- Loop Unrolling

Other Loop Optimizations

Other Loop Optimizations

- Scalar replacement
- Loop Interchange
- Loop Fusion
- Loop Distribution
- Loop Skewing
- Loop Reversal

Loop Invariant Code Motion

- Build the SSA graph
- Need *semi-pruned* insertion of ϕ -nodes:

If two non-null paths $x \rightarrow^+ z$ and $y \rightarrow^+ z$ converge at node z , and nodes x and y contain assignments to t (in the original program), then a ϕ -node for t must be inserted at z (in the new program)

and t must be live across some basic block

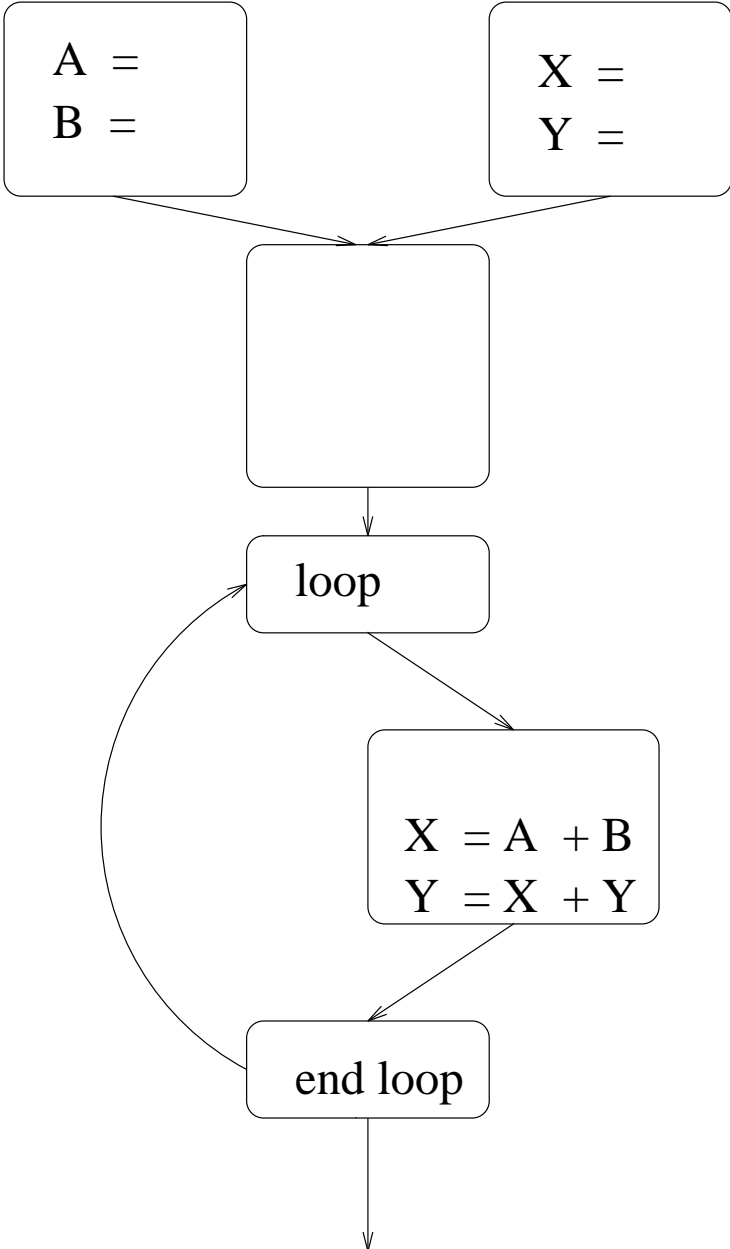
Simple test:

If, for a statement $s \equiv [x \leftarrow y \otimes z]$, none of the operands y, z refer to a ϕ -node or definition inside the loop, then

Transform:

assign the invariant computation a new temporary name, $t \leftarrow y \otimes z$, move it to the loop pre-header, and assign $x \leftarrow t$.

Loop Invariant Code Motion: Example I



Loop Invariant Code Motion

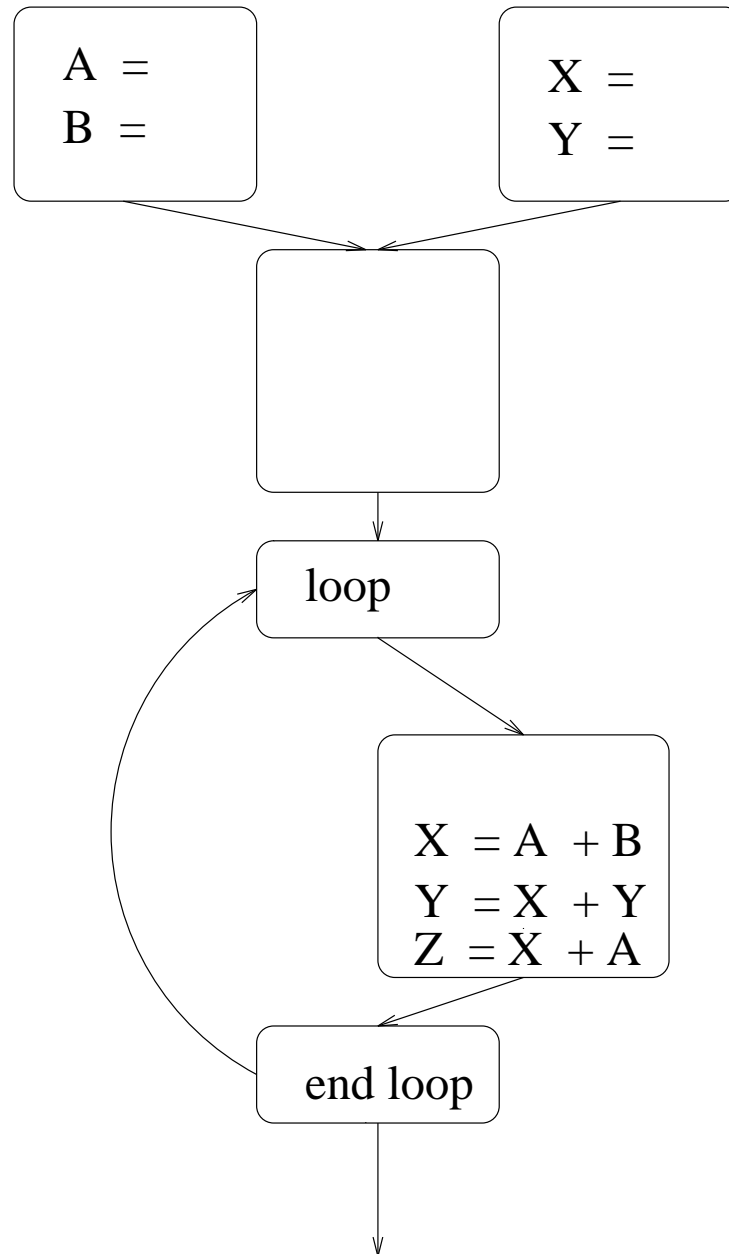
More invariants

Start at loop entry point:

Test: If operands point to definitions inside loop, and those definitions are a function of loop invariants (*recursive definition*)

Transform: as before for each invariant

Loop Invariant Code Motion: Example II



Induction Variable Recognition

- What is a loop induction variable?
- Why might we want to detect one?

```
 $i \leftarrow 0$   
while  $i < 10$  do  
     $i \leftarrow i + 1$   
end
```

Simplest Method: Pattern match for $i \leftarrow i + c$ in loop and ensure no other definition of i in loop.

Does not catch all loop induction variables.

Taxonomy of Induction Variables

1. A *basic* induction variable is a variable i

- whose only definition within the loop is an assignment of the form $i \leftarrow i \pm c$, where c is loop invariant.

2. A *mutual* induction variable i' is

- defined *once* within the loop, and its value is a linear function of some other induction variable i such that

$$i' \leftarrow i \otimes c_1 \pm c_2$$

where \otimes is one of \times or $/$, and c_1, c_2 are loop invariant.

3. The *family* of a basic induction variable i :

- the set of mutual induction variables on i

Optimistic Induction Variable Recognition

```
 $IV \leftarrow \{\}$   
foreach statement  $s$  in loop do  
  if  $s \equiv [i \leftarrow x \pm c] \wedge (c \text{ is loop invariant})$   
     $IV \leftarrow IV \cup \{i\}$   
  elsif  $s \equiv [i \leftarrow x \otimes c] \wedge c \text{ is loop invariant}$   
     $IV \leftarrow IV \cup \{i\}$   
  end  
end  
do  
   $changed \leftarrow false$   
  foreach  $s \equiv [i \leftarrow \dots] \in IV$  do  
    if  $\exists u \in Uses(s) : u \notin IV$   
       $IV \leftarrow IV - \{i\}$   
       $changed \leftarrow true$   
    end  
  end  
while  $changed$ 
```

Finds linear induction variables and catches mutual induction variables.

Optimistic Induction Variables

$i \leftarrow 0$

$k \leftarrow 0$

do

$j \leftarrow k + 1$

$k \leftarrow j + 2$

$i \leftarrow i \times 2$

end

Loop Induction Variables with SSA

- Build the SSA graph
- Going from the innermost to the outermost loop
- Find cycles in the SSA graph

Each cycle *may* be for a *basic* induction variable

if the variable in the cycle is a function of loop invariants and its value on the current iteration

(ie, its ϕ is a function of an *initialized* variable and an instance of v in the cycle)

- Other induction variables can depend on basic induction variables.

Loop Induction Variables: Example I

```
 $i \leftarrow 1$   
do  
    ... ( $i$ ) ...  
     $i \leftarrow i + 1$   
    ... ( $i$ ) ...  
end
```

```
 $i_1 \leftarrow 1$   
do  
     $i_2 \leftarrow \phi(i_1, i_3)$   
    ... ( $i_2$ ) ...  
     $i_3 \leftarrow i_2 + 1$   
    ... ( $i_3$ ) ...  
end
```

Loop Induction Variables with SSA

How to determine: *If the variable(s) in the cycle is(are) a function of loop invariants and its value on the current iteration:*

- The ϕ -node in the cycle will take one definition from inside the loop and one from outside the loop (*assuming ϕ -nodes with only two inputs*)
- The definition inside the loop will be part of the cycle and will get one operand from the ϕ -node and any others will be loop invariant
- For linear induction variables the operator will be addition, subtraction, or unary minus

Loop Induction Variables: Example II

$i \leftarrow 3$
 $m \leftarrow 0$
do

$j \leftarrow 3$
 $i \leftarrow i + 1$
 $l \leftarrow m + 1$
 $m \leftarrow l + 2$
 $j \leftarrow i + 2$
 $k \leftarrow 2 \times j$

end

\Rightarrow

$i_1 \leftarrow 3$
 $m_1 \leftarrow 0$
do
 $i_2 \leftarrow \phi(i_1, i_3)$
 $m_2 \leftarrow \phi(m_1, m_3)$
 $j_1 \leftarrow 3$
 $i_3 \leftarrow i_2 + 1$
 $l_1 \leftarrow m_2 + 1$
 $m_3 \leftarrow l_1 + 2$
 $j_2 \leftarrow i_3 + 2$
 $k_1 \leftarrow 2 \times j_2$

end