

## Reuse Optimization

### Last time

- Dead code elimination
- Common subexpression elimination (CSE)
- Copy propagation
- Simple constants

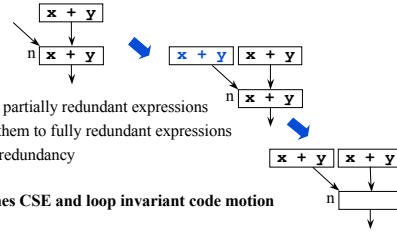
### Today

- Partial redundancy elimination (PRE)

## Partial Redundancy Elimination (PRE)

### Partial Redundancy

- An expression (e.g.,  $x+y$ ) is **partially redundant** at node  $n$  if **some** path from the entry node to  $n$  evaluates  $x+y$ , and there are no definitions of  $x$  or  $y$  between the last evaluation of  $x+y$  and  $n$



### Elimination

- Discover partially redundant expressions
- Convert them to fully redundant expressions
- Remove redundancy

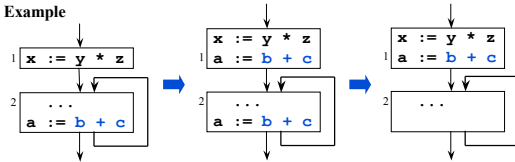
### PRE subsumes CSE and loop invariant code motion

## Loop Invariance Example

### PRE removes loop invariants

- An invariant expression is partially redundant
- PRE converts this partial redundancy to full redundancy
- PRE removes the redundancy

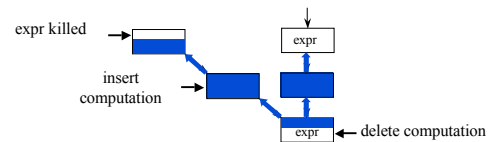
### Example



## Implementing PRE [lazy code motion Knoop 92, dragon 2007]

### Big picture

- Use global analysis (data-flow analysis) to discover where partial redundancy can be converted to full redundancy
- Global analysis also determines latest possible point to create redundancy
- Insert code and remove redundant expressions
- As in textbook, assuming one statement per basic block



## Local Properties

An expression is locally **available (or in  $e\_gen[b]$ )** in block  $b$  if it is computed at least once and its operands are not modified after its last computation in  $b$ .

An expression is locally **anticipated** if it is computed at least once and its operands are not modified before its first evaluation

An expression is locally **used (or in  $e\_use[b]$ )** in block  $b$  if it is computed at least once. With only one statement per block, **anticipated =  $e\_use[b]$**

An expression is locally **killed (or in  $e\_kill[b]$ )** in block  $b$  if any of its operands are defined in  $b$ .

**Example**

$b := b + c$	Available: $\{\}$
	Anticipated: $\{b + c\}$
	$e\_use$ : $\{b + c\}$
	$e\_kill$ : $\{b + c, b*a, \dots\}$

CS553 Lecture

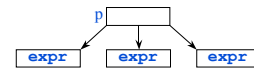
Reuse Optimization: PRE

5

## Global Anticipability

### Intuition

– If  $e$  is globally anticipated at  $p$ , then an evaluation of  $e$  at  $p$  will make the next evaluation of  $e$  redundant along all paths from  $p$



### Flow Functions

$$\text{anticipated\_out}[n] = \bigcap_{s \in \text{succ}[n]} \text{anticipated\_in}[s]$$

$$\text{anticipated\_in}[n] = e\_use[n] \cup (\text{anticipated\_out}[n] - e\_kill[n])$$

CS553 Lecture

Reuse Optimization: PRE

6

## Global Availability for PRE

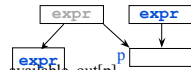
### Intuition

- Global availability for PRE is almost the same as Available Expressions, except it depends on the results of global anticipated expressions
- If  $e$  is globally available at  $p$ , then an evaluation at  $p$  will create redundancy along all paths start at  $p$

### Flow Functions

$$\text{available\_in}[n] = \bigcap_{p \in \text{pred}[n]} \text{available\_out}[p]$$

$$\text{available\_out}[n] = (\text{anticipated\_in}[n] - e\_kill[n]) \cup (\text{available\_in}[n] - e\_kill[n])$$



CS553 Lecture

Reuse Optimization: PRE

7

## Earliest

### Intuition

- The earliest place an expression is anticipated, but not globally available.
- Does not require iterative data-flow analysis. Just requires one pass over all statements.
- Could place an expression generation statement at the beginning of any block  $b$  where expression is in  $\text{earliest}[b]$

### Function

$$\text{earliest}[n] = \text{anticipated\_in}[n] - \text{available\_in}[n]$$

CS553 Lecture

Reuse Optimization: PRE

8



## Global Used Expressions

### Intuition

- If  $e$  is globally used at  $p$ , then an evaluation of  $e$  at  $p$  will be used again along some path starting at  $p$ .
- If an expression is not in the  $used\_out[b]$ , then a computation of the expression should not be put at the beginning of block  $b$ , even if  $e$  is in  $latest[b]$ .
- "Liveness analysis for expressions."

### Flow Functions

$$used\_out[n] = \bigcup_{s \in succ[n]} used\_in[s]$$

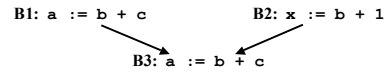
$$used\_in[n] = (e\_use[n] - latest[n]) \cup (used\_out[n] - latest[n])$$

CS553 Lecture

Reuse Optimization: PRE

13

## Example



	B1	B2	B3
<b>e use</b>	{b+c}	{b+1}	{b+c}
<b>e kill</b>	{}	{}	{}
<b>anticipated_out</b>	{b+c}	{b+c}	{}
<b>anticipated_in</b>	{b+c}	{b+c, b+1}	{b+c}
<b>available_in</b>	{}	{}	{b+c}
<b>available_out</b>	{b+c}	{b+c, b+1}	{b+c}
<b>earliest</b>	{b+c}	{b+c, b+1}	{}
<b>postponable_in</b>	{}	{}	{}
<b>postponable_out</b>	{}	{b+c}	{}
<b>latest</b>	{b+c}	{b+c, b+1}	{}
<b>used_out</b>	{b+c}	{b+c}	{}
<b>used_in</b>	{}	{}	{b+c}

CS553 Lecture

Reuse Optimization: PRE

14

## PRE Summary

### Algorithm

- Insert an empty block along all edges entering a block with more than one predecessor.
- Calculate latest and  $used\_out$  sets.
- For each expression  $e$ 
  - create a temporary  $t$  to store  $e$
  - for all blocks where  $e$  is in  $latest[b]$  and  $used\_out[b]$ , add  $t=e$  to beginning of block
  - for all blocks where  $e$  is in  $(e\_use[b]$  and (not  $latest[b]$  or  $used\_out[b]$ )), replace original  $e$  with  $t$

### What's so great about PRE?

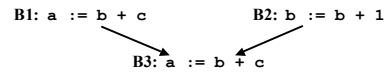
- A modern optimization that subsumes earlier ideas
- Composes several simple data-flow analyses to produce a powerful result
  - Finds earliest and latest points in the CFG at which an expression is anticipated

CS553 Lecture

Reuse Optimization: PRE

15

## Another Example



	B1	B2	B3
<b>e use</b>	{b+c}	{b+1}	{b+c}
<b>e kill</b>	{}	{b+c, b+1}	{}
<b>anticipated_out</b>	{b+c}	{b+c}	{}
<b>anticipated_in</b>	{b+c}	{b+1}	{b+c}
<b>available_in</b>	{}	{}	{}
<b>available_out</b>	{b+c}	{}	{b+c}
<b>earliest</b>	{b+c}	{b+1}	{b+c}
<b>postponable_in</b>	{}	{}	{}
<b>postponable_out</b>	{}	{}	{}
<b>latest</b>	{}	{b+1}	{b+c}
<b>used_out</b>	{}	{}	{}
<b>used_in</b>	{}	{}	{}

CS553 Lecture

Reuse Optimization: PRE

16

## Next Time

---

### Assignments

- HW1 has been posted

### Lecture

- Control flow
- Loops
- Dominators