Reuse Optimization

Last time

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

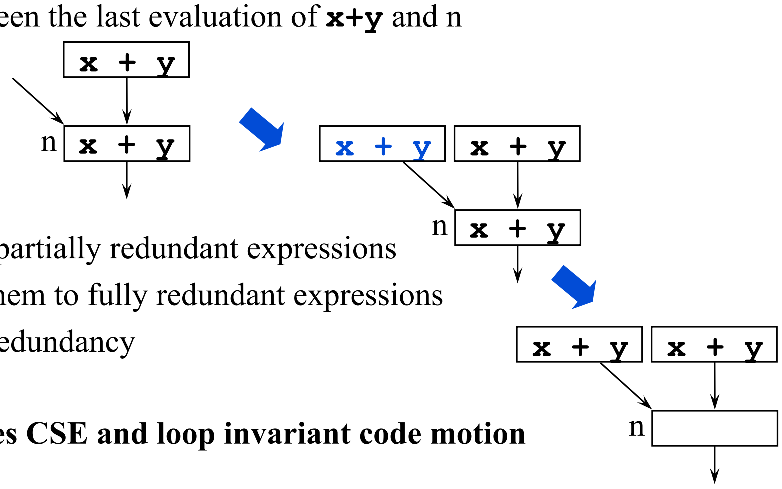
Today

– 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

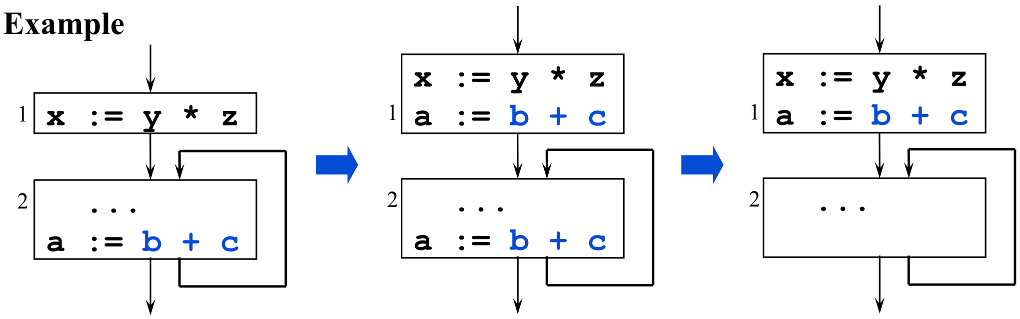
PRE subsumes CSE and loop invariant code motion

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

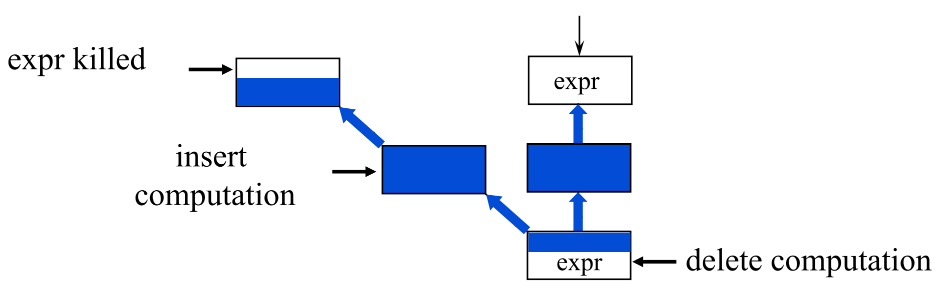
Example



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

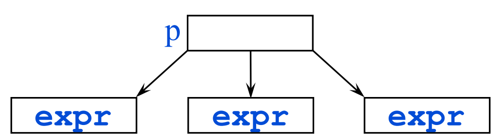
```
b := b + c
```

Available:	{}
Anticipated:	{`b + c`}
e_use:	{`b + c`}
e_kill:	{`b + c, b*a, ...`}

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

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

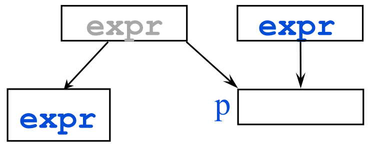
$$\text{anticipated\_in}[n] = \text{e\_use}[n] \cup (\text{anticipated\_out}[n] - \text{e\_kill}[n])$$

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 starting at p



Flow Functions

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

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

Earliest

Intuition

– The earliest place an expression is anticipated in, but not globally available in.

– 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 earliest[b]

Function

earliest[n] = anticipated_in[n] - available_in[n]

Example

B1: a := b + c   B2: x := b + 1

B3: a := b + c

	B1	B2	B3
e_use	{b+c}	{b+1}	{b+c}
e_kill	{}	{}	{}
anticipated_out	{b+c}	{b+c}	{}
anticipated_in	{b+c}	{b+c, b+1}	{b+c}
available_in	{}	{}	{b+c}
available_out	{b+c}	{b+c, b+1}	{b+c}
earliest	{b+c}	{b+c, b+1}	{}
postponable_in			
postponable_out			
latest			
used_out			
used_in			

Postponable Expressions

Intuition

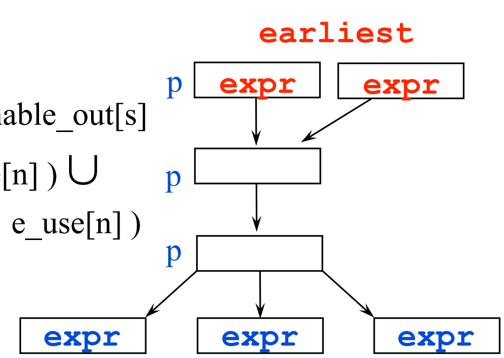
– Calculates all possible program points starting from earliest in which an expression can be computed while preserving semantics and not introducing more redundancy.

– By computing an expression at the latest possible postponable point, the live range of the temporary that holds expression value is reduced.

Flow Functions

postponable_in[n] = $\bigcap_{s \in pred[n]}$ postponable_out[s]

postponable_out[n] = (earliest[n] - e_use[n]) $\cup$ (postponable_in[n] - e_use[n])



Latest

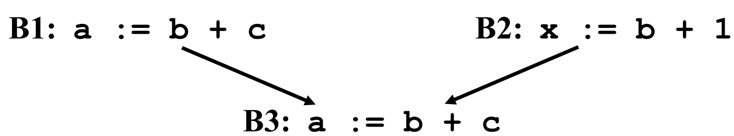
Intuition

- Last block the computation for which the expression computation can be postponed.
- If an expression is in latest[b] that indicates that the last point the expression can be computed is at the beginning of block b.
- Requires only one visit to each statement.

Flow Function

$$\text{latest}[n] = (\text{earliest}[n] \cup \text{postponable\_in}[n]) \cap (\text{e\_use}[n] \cup \text{not}(\bigcap_{s \in succ[n]} (\text{earliest}[s] \cup \text{postponable\_in}[s])))$$

Example



	B1	B2	B3
e_use	{b+c}	{b+1}	{b+c}
e_kill	{}	{}	{}
anticipated_out	{b+c}	{b+c}	{}
anticipated_in	{b+c}	{b+c, b+1}	{b+c}
available_in	{}	{}	{b+c}
available_out	{b+c}	{b+c, b+1}	{b+c}
earliest	{b+c}	{b+c, b+1}	{}
postponable_in	{}	{}	{}
postponable_out	{}	{b+c}	{}
latest	{b+c}	{b+c,b+1}	{}
used_out			
used_in			

Global Used Expressions

Intuition

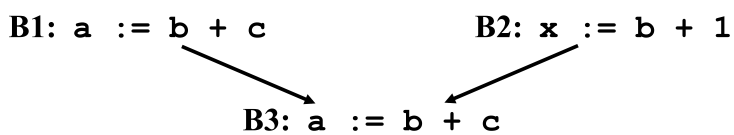
- If e is globally used at p, then an evaluation of e at p will be used again along some path starting at p.
- If an expression is not in the used_out[b], then a computation of the expression should not be put at the beginning of block b, even if e is in latest[b].
- “Liveness analysis for expressions.”

Flow Functions

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

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

Example



	B1	B2	B3
e_use	{b+c}	{b+1}	{b+c}
e_kill	{}	{}	{}
anticipated_out	{b+c}	{b+c}	{}
anticipated_in	{b+c}	{b+c, b+1}	{b+c}
available_in	{}	{}	{b+c}
available_out	{b+c}	{b+c, b+1}	{b+c}
earliest	{b+c}	{b+c, b+1}	{}
postponable_in	{}	{}	{}
postponable_out	{}	{b+c}	{}
latest	{b+c}	{b+c,b+1}	{}
used_out	{b+c}	{b+c}	{}
used_in	{}	{}	{b+c}

PRE Summary

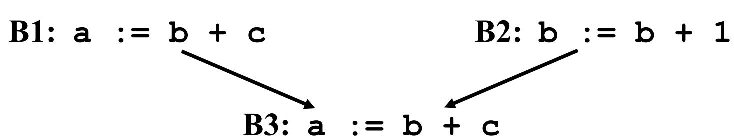
Algorithm

- 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

Another Example



	B1	B2	B3
e_use	{b+c}	{b+1}	{b+c}
e_kill	{}	{b+c, b+1}	{}
anticipated_out	{b+c}	{b+c}	{}
anticipated_in	{b+c}	{b+1}	{b+c}
available_in	{}	{}	{}
available_out	{b+c}	{}	{b+c}
earliest	{b+c}	{b+1}	{b+c}
postponable_in	{}	{}	{}
postponable_out	{}	{}	{}
latest	{}	{b+1}	{b+c}
used_out	{}	{}	{}
used_in	{}	{}	{}

Next Time

Assignments

– HW1 due tomorrow

Midterm on Tuesday

– Email questions

– Do suggested exercises