Partial Redundancy

15-745 Optimizing Compilers
Spring 2006

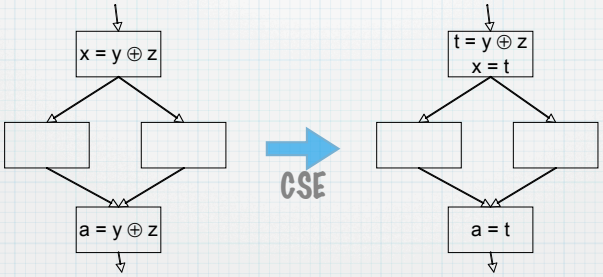
Peter Lee

Reminders

* Task 2 is out
  * mainly: identify loops and check hoisting conditions
* Read 13.3 (partial redundancy elimination)

Global CSE

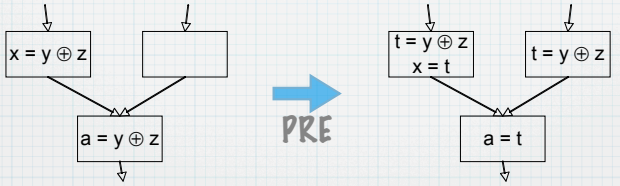
* We have already seen common subexpression elimination



A form of **redundancy elimination**

Partial redundancies

* For CSE we use available-expressions
* But many redundancies are **partial**



computation is redundant on some but not all paths

partial redundancy elimination

Available expressions

* Consider expression **e** of the form, **y ⊕ z**
* **e** is **available** at node **n** if it definitely has already been evaluated, without having been killed in the mean time
  * also called **up-safe**

AE analysis

* $gen_{ae}(n)$ = exprs evaluated in n
* $kill_{ae}(n)$ = exprs whose vars are modified in n
* $in_{ae}(n)$ is initially $\perp$, for all n

$$in_{ae}(n) = \bigwedge_{p \in pred(n)} out_{ae}(p)$$

$$out_{ae}(n) = (in_{ae}(n) - kill_{ae}(n)) \vee gen_{ae}(n)$$

Alternative formulation of AE

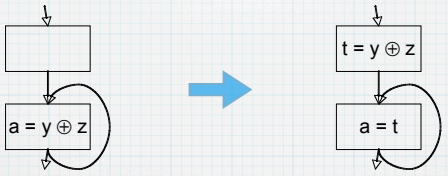
* trans(n) = exprs whose vars are **not** modified in n

$$in_{ae}(n) = \bigwedge_{p \in pred(n)} out_{ae}(p)$$

$$out_{ae}(n) = (in_{ae}(n) \cap trans(n)) \vee gen_{ae}(n)$$

Loop invariants

* Note that loop-invariant computations are examples of partially redundant computations



PRE performs CSE and loop-invariant hoisting, in addition to optimizing partial redundancies

PRE

* The idea of partially redundant computations and its connection to loop-invariant hoisting and CSE is an old idea, due to Morel & Renvoise in 1979
* Knoop, Ruthing & Steffen provided the first widely accepted practical algorithm for PRE in their 1992 PLDI paper (later TOPLAS 1994)
  * "lazy code motion", aka "optimal code motion"

Dataflow analysis

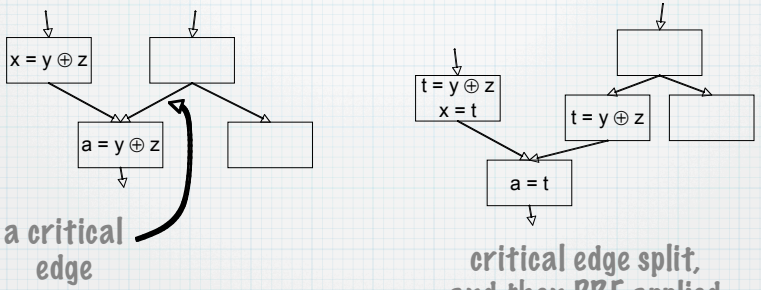
* Although an old idea, PRE was adopted into practice and actively studied only recently
* This is due in part to the difficulty in explaining the analysis, which is surprising since it involves only dataflow over BV^n

How to do PRE

Step 1: Preparing for PRE

* Just as we needed to create pre-header nodes for loop-invariant hoisting, for PRE we will need to split critical edges in the flowgraph
* A **critical edge** connects a node with >1 successors to a node with >1 predecessor

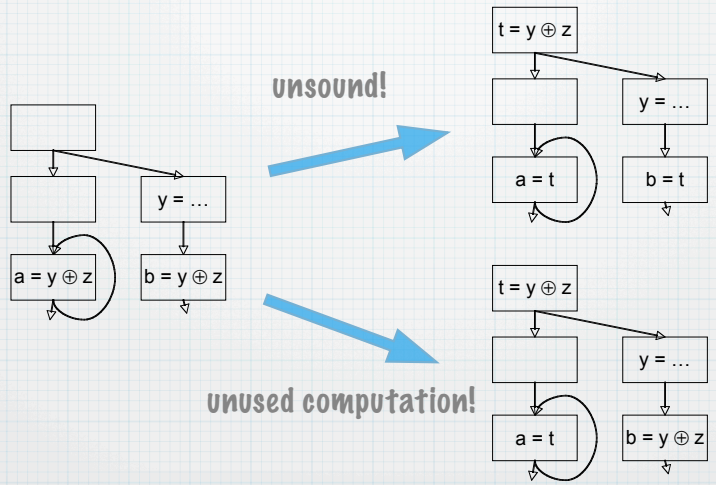
Splitting critical edges



Step 2: Safe code motion

* As all of our examples show, PRE involves hoisting of computations
* We can start by computing in which blocks each expression can be safely hoisted
* By "safe," we mean that hoisting would definitely **not** create any unused values

Hoist up, but not too far!



Very busy expressions

* Consider expression **e** of the form, **y ⊕ z**
* **e** is **very busy** at node **n** if the result of its evaluation at the beginning of **n** would definitely be used before either **y** or **z** is changed
  * a "backward AE"
  * also called **down-safe** or **anticipatable**

VBE analysis

* $gen_{vbe}(n) = gen_{ae}(n)$
* $out_{vbe}(n)$ is initially $\perp$, for all n

$$in_{vbe}(n) = (out_{vbe}(n) \cap trans(n)) \vee gen_{vbe}(n)$$

$$out_{vbe}(n) = \bigwedge_{s \in succ(n)} in_{vbe}(s)$$

Step 3: Choosing a node

* The VBE analysis tells us where it is safe to evaluate each expression
* So, if an expression can be safely evaluated at multiple places, how do we choose among them?

Earliest safe nodes

* To minimize the number of computations, we want to choose, for each expression, the earliest safe node
* Earliestness:
  * **e** is earliest into **n** if no block from **entry** to **n** both evaluates **e** and produces the same value as evaluating **e** at **n**

Earliest

* Essentially:
  * the entry node, if it is safe,
  * a node with a predecessor that kills, or
  * a node with a predecessor that isn't safe

Earliest analysis

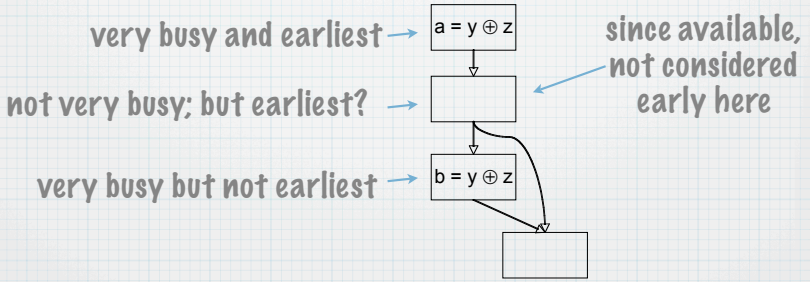
$in_{early}(entry)$ initially $\top$

$in_{early}(n) = \bigvee_{p \in pred(n)} out_{early}(p)$

$out_{early}(n) = kill_{ae}(n) \vee (in_{early}(n) - in_{vbe}(n))$

Alternative formulation

* Some authors want to take into account availability



Earliest analysis, alternatively

$in_{early}(entry)$ initially $\top$

$in_{early}(n) = \bigvee_{p \in pred(n)} out_{early}(p)$

$out_{early}(n) = kill_{ae}(n) \vee (in_{early}(n) - (in_{vbe}(n) \vee in_{ae}(n)))$

Naive PRE

* We can now perform a naive PRE:
  * if $y \oplus z$ is early in n:
    * insert $t = y \oplus z$ at the beginning of n
      * new temp t
    * and replace occurrences of $y \oplus z$ with t

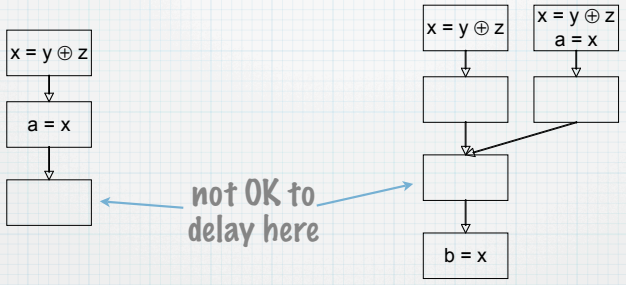
Step 4: Being register-friendly

* The earliest analysis is a correct PRE with respect to eliminating redundant computations
* However, moving computations as early as possible maximizes the live range of values, thereby increasing the demand for registers to store them
* This leads to more register spilling

Delaying computations

* The conflict between register allocation and redundancy elimination is a general problem
* While not solving the problem, we can improve the situation by delaying moved computations to be as late as possible
* This is known as **lazy code motion**

Delaying examples



Delaying computations

* To figure out to what point we can delay a computation, we follow a similar strategy to the earliest analysis:
  * identify where it is OK to delay,
  * then choose the latest of the OK nodes

Delayedness analysis

$$in_{delay}(entry) = in_{early}(entry) \wedge in_{vbe}(entry)$$

$$in_{delay}(n) = (in_{early}(n) \wedge in_{vbe}(n)) \vee \bigwedge_{p \in pred(n)} out_{delay}(p)$$

$$out_{delay}(n) = in_{delay}(n) - gen_{ae}(n)$$

Choosing the latest node

* With delay information, we can (finally!) choose the latest node for each expression
  * node that uses the expression, or
  * node that has a successor n for which the expression is not in delay(n)

Latestness

$$late(n) = in_{delay}(n) \wedge (gen_{ae}(n) \vee \bigvee_{s \in succ(n)} \neg in_{delay}(s))$$

PRE

* Compute late(n), for each node n
* For expression **e**, insert statement
  * **t = e**
  * where **t** is new, in beginning of n
* Replace other occurrences of **e** with **t**

Next time

* SSA form