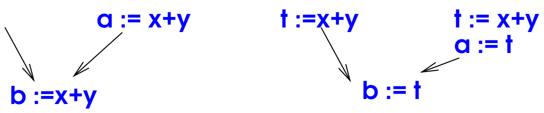
Partial Redundancy Elimination

CSE and Loop-Invariant Code Motion, and more

A computation e is partially redundant ***if e is calculated more than once on some path in the CFG without change of values.***

Aim: Move partially redundant computations to optimal point, and eliminate totally redundant computations

Ex.

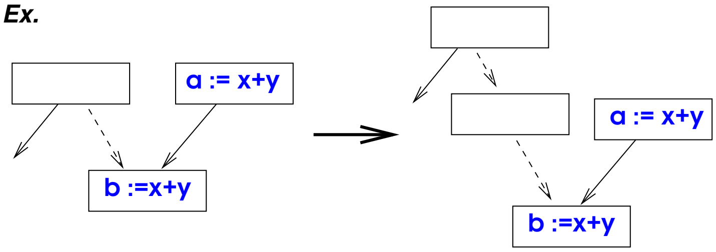


CSE not applicable since expression not available on ***all*** paths

Good idea?

Where move code?

Ex.



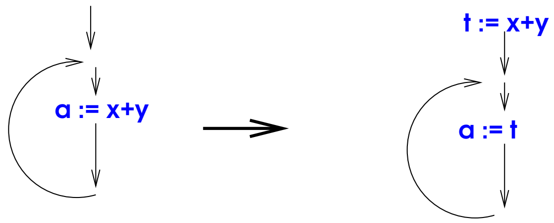
Critical edge: From node with more than 1 successor
To node with more than 1 predecessor

Split critical edge first, creating new basic block

Use new basic block for code placement, as before

What about loops?

Ex.



Subsumes loop-invariant code motion

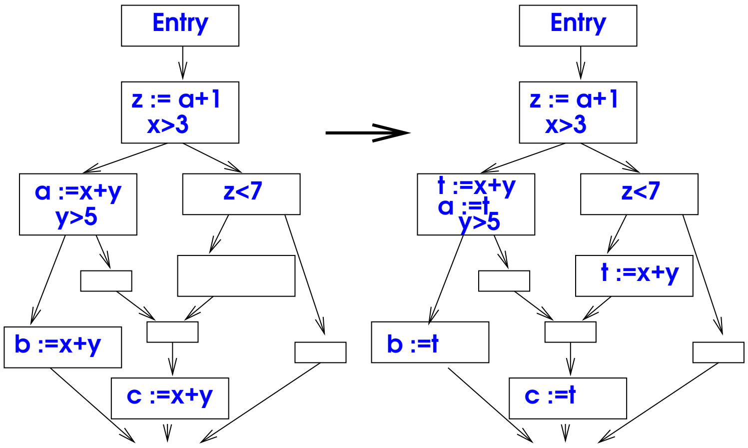
Overview of PRE

1. Solve data flow problem(s) to discover location of partially redundant expressions
2. Solve data flow problem(s) to find "optimal" points where inserting a copy makes partially redundant exp. fully redundant
3. Insert code and delete fully redundant copies

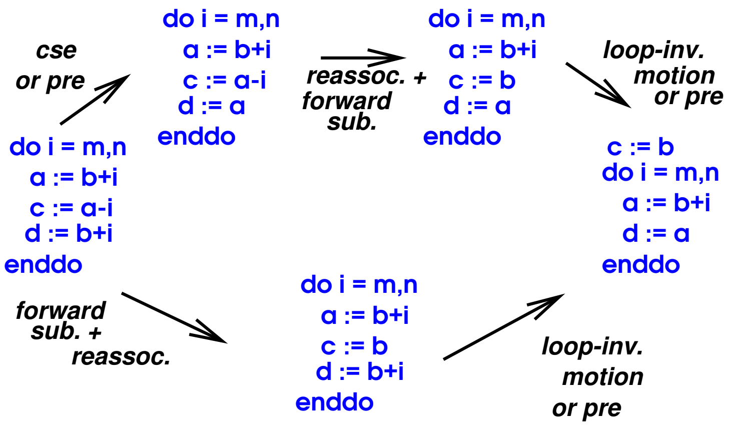
Original formulation used bi-directional data flow analyses

Newer approach solves as sequence of forward, backward problems

Example



Interaction with other optimizations



Code Hoisting

Application of Very Busy Expressions

Def. ***An expression e is*** very busy ***at B if on all paths from B, e is evaluated before any of its operands are defined.***

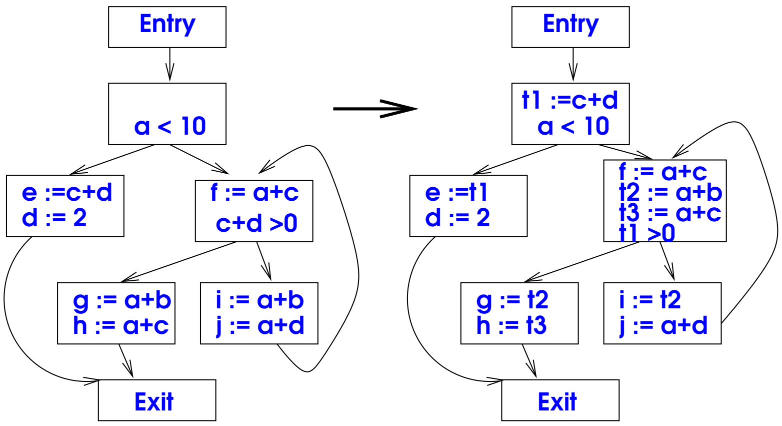
B is potential point to hoist code

Want earliest such point

Beneficial?

May not always improve execution time

Example



Induction Variables & Strength Reduction

Induction variable: ***variable whose successive values form an arithmetic progression in loop***

```
Ex. do i = 1,99                    t := -1
       a(i) :=2*i-1       -->      do i = 1,99
    enddo                             t:=t1+2
                                      a(i) := t
                                   enddo
```

i is inductive
2*i-1 is inductive

Strength Reduction

Replaces expensive ops (like *) with less expensive ops (like +)

Loop Peeling

Remove 1 or more iterations from beginning/end of loop

```
                      i = 1            a(1) := a(1) + a(1)
Ex. do i = 1,100               -->     do i = 2,100
       a(i) := a(i) + a(1)                a(i) := a(i) + a(1)
    enddo                              enddo
                    i > 1
```

Beneficial?

Always possible?

Index Set Splitting

Split index set 1,...,N into multiple sets

Ex.
```
do i = 1,100
    a(i) :=b(i) +c(i)
    if (i>50) then
        d(i) := a(i) + a(i-10)
enddo
```
→
```
do i = 1,50
    a(i) :=b(i) +c(i)
enddo
do i = ,51,100
    a(i) :=b(i) +c(i)
    d(i) := a(i) + a(i-10)
enddo
```

Beneficial?

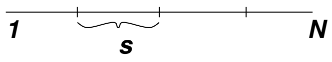
Always possible?

Strip Mining

Turn 1 loop into 2:

Outer: by strip size s

Inner: inside each strip



Ex.
```
do i = 1,100
    a(i+20) :=a(i) +b
enddo
```
→
```
do is = 1,100,20
    do i = is, is+19
        a(i+20) :=a(i) +b
    enddo
enddo
```

Always possible?

Beneficial?

Loop Unrolling

Copy body of loop k times, substituting successive indices and adjusting outer loop accordingly

```
Ex.  do i = 1,99                      .  do i = 1,99,3
          a(i) := b(i) +c                     a(i) := b(i) +c
                                 -->          a(i+1) := b(i+1) +c
                                              a(i+2) := b(i+2) +c
        enddo                                enddo
```

Always possible?

Beneficial?