Partial Dead Code Elimination

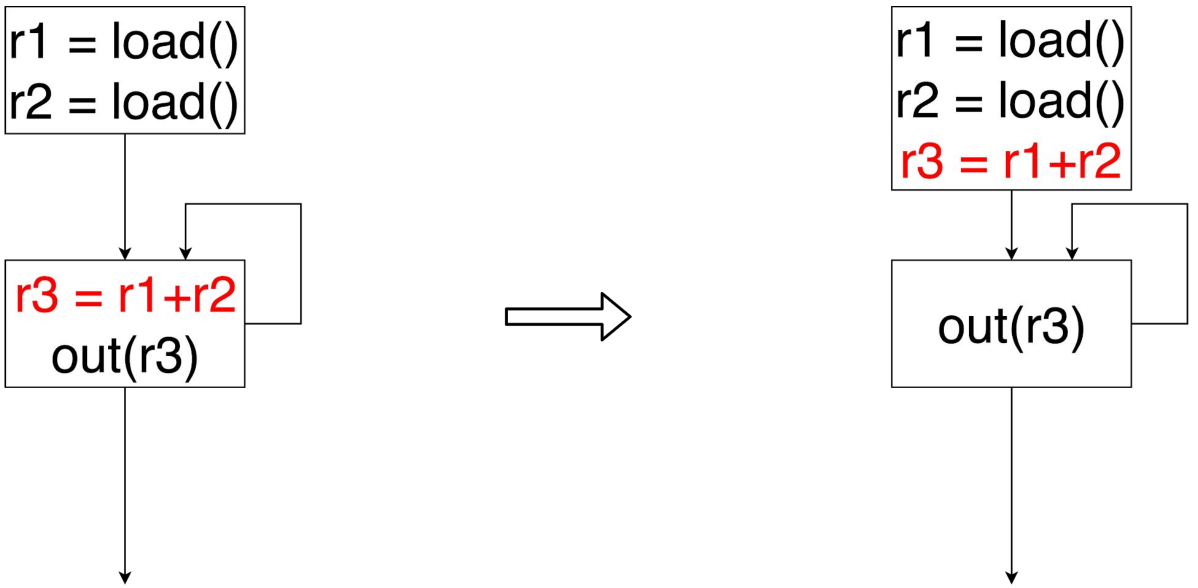
Written by Jens Knoop, Oliver Rüthing & Bernhard Steffen

Presented by Yian Zhu, Tengda Tang & Chess Luo

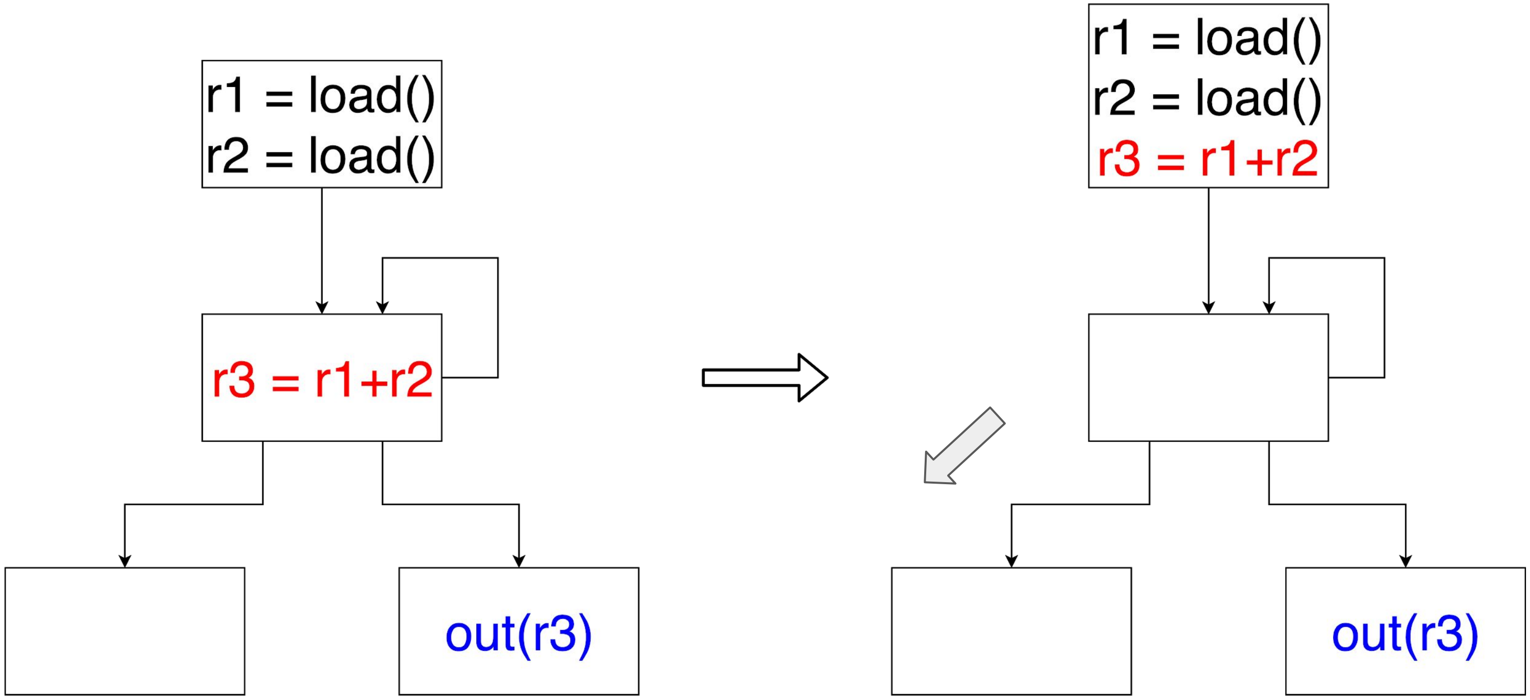
Partial Dead Code Elimination - Quick Facts

- Machine independent
- IR level optimization
- Optimize for speed by reducing dynamic operation count

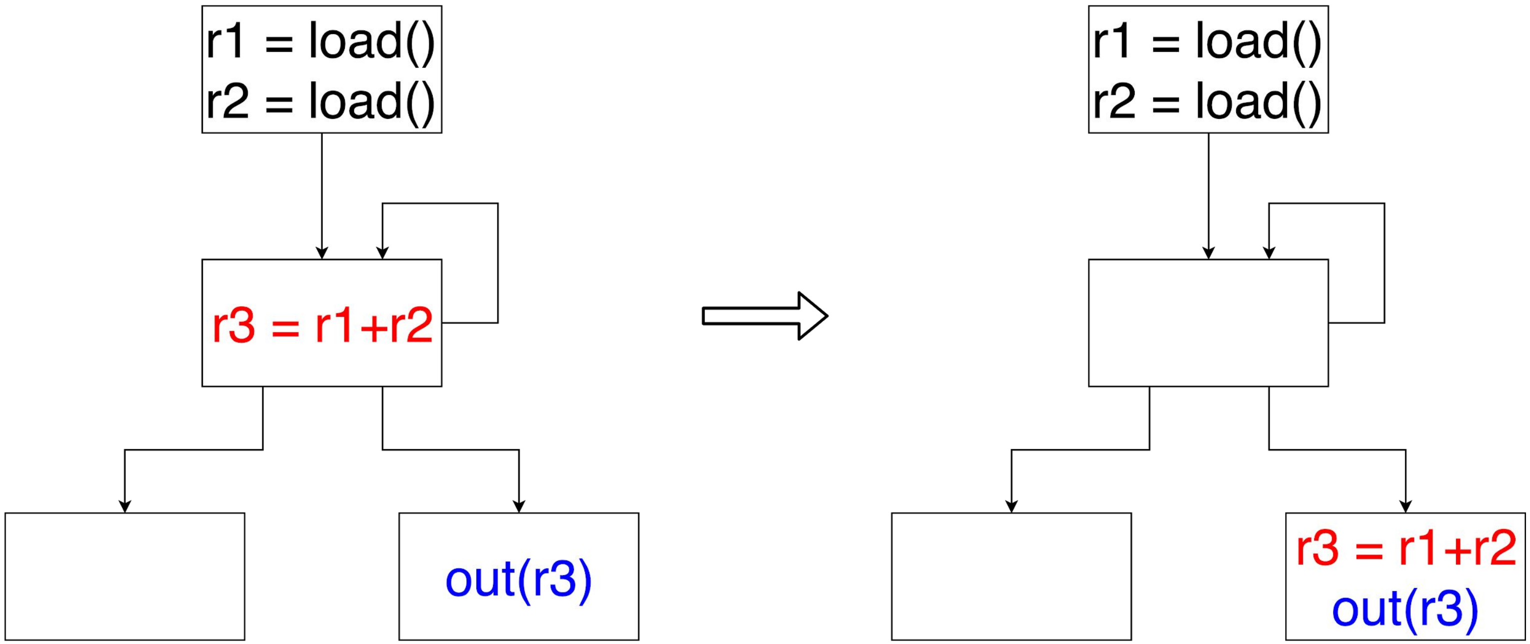
Loop Invariant Code Motion



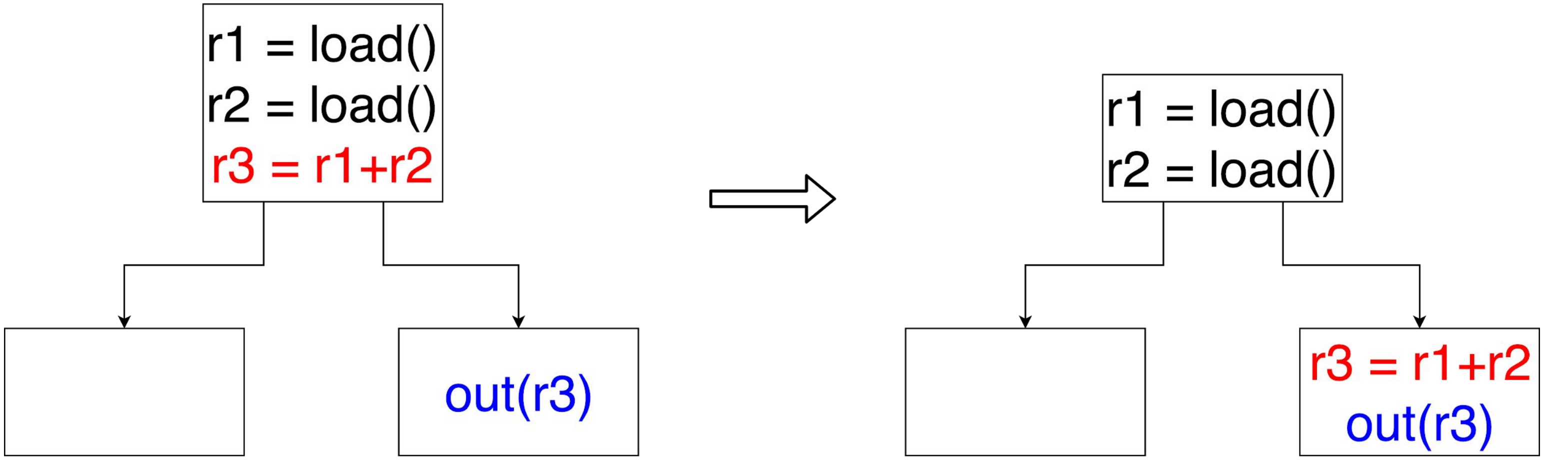
What if...



Partially Dead Code Elimination

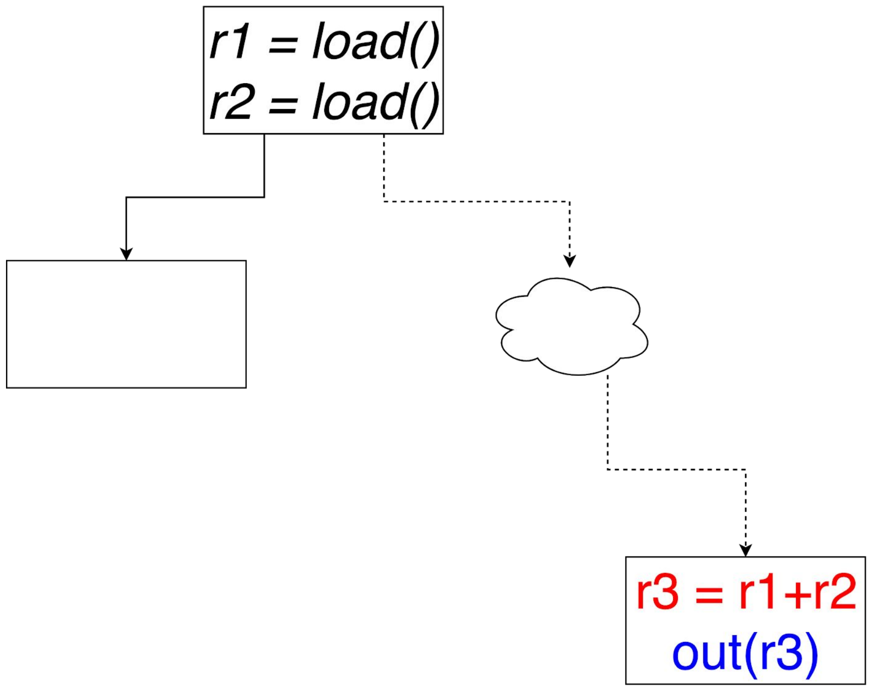


Partial Dead Code Elimination - Simple Case



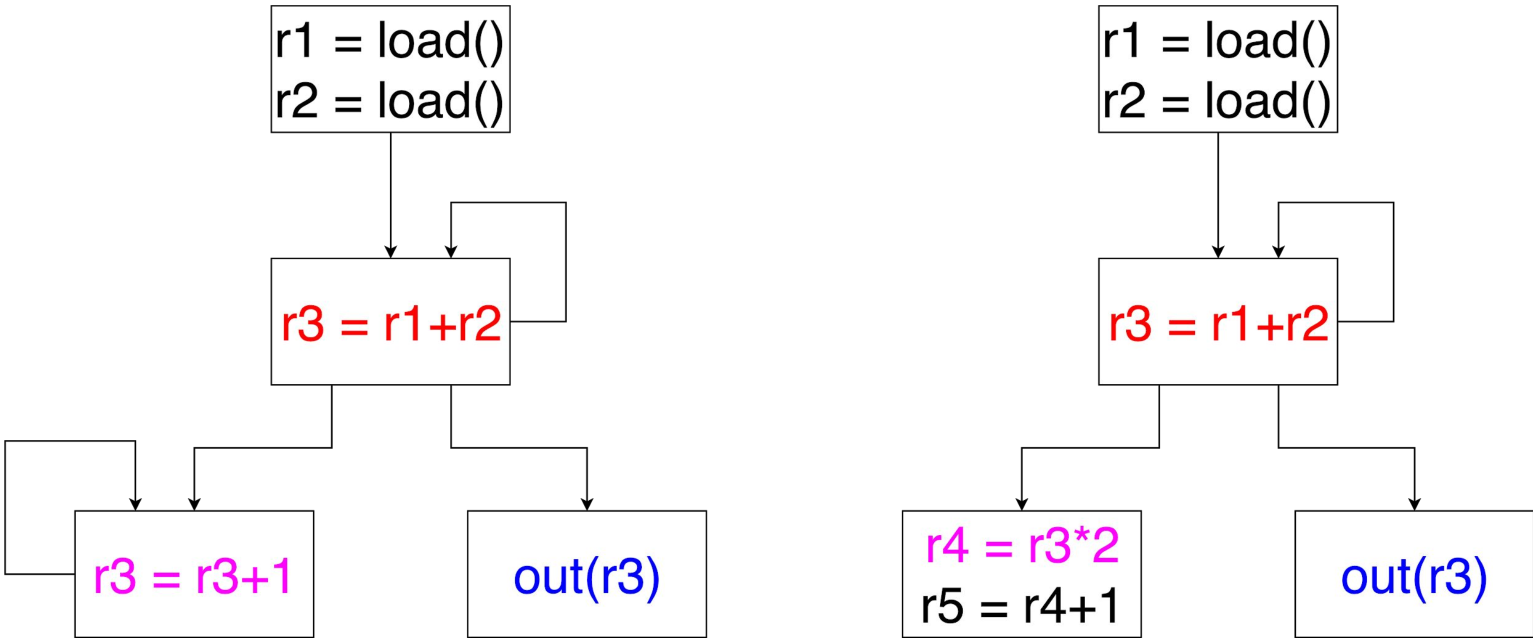
Partial Dead Code Elimination - Real World

1. Move the instruction through all kinds of control flow to reach its “live” branch



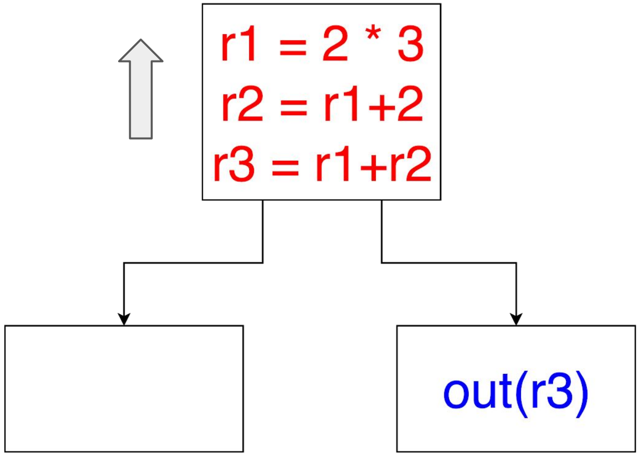
Partial Dead Code Elimination - Real World

1. Move the instruction through all kinds of control flow to reach its “live” branch
2. Have to deal with “faint” code situation
    - Either the left hand side has no usage at all or it is killed by other assignment before any usage (the definition of dead code)
    - Or the left hand side is only used by other faint code

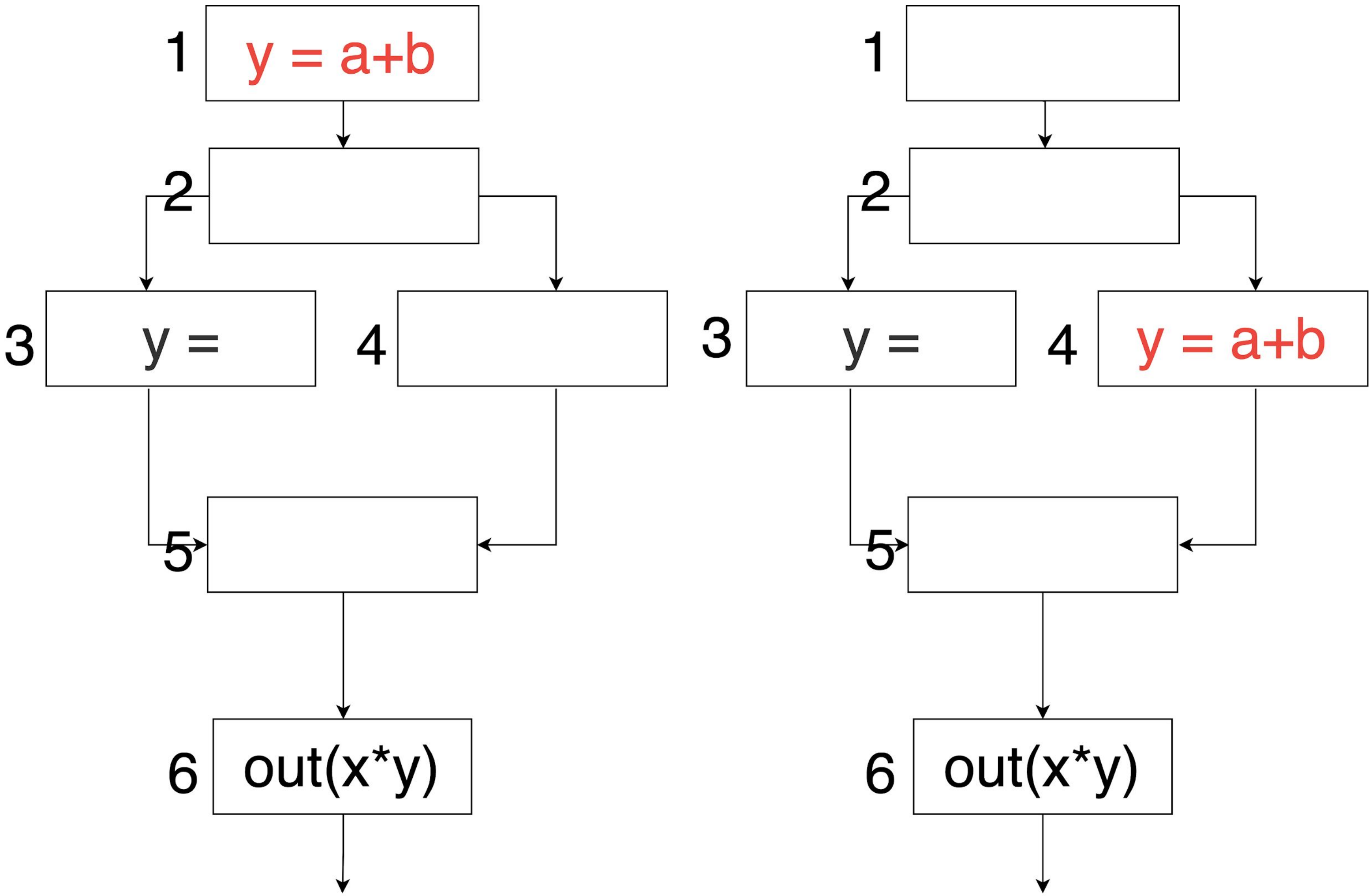


Partial Dead Code Elimination - Real World

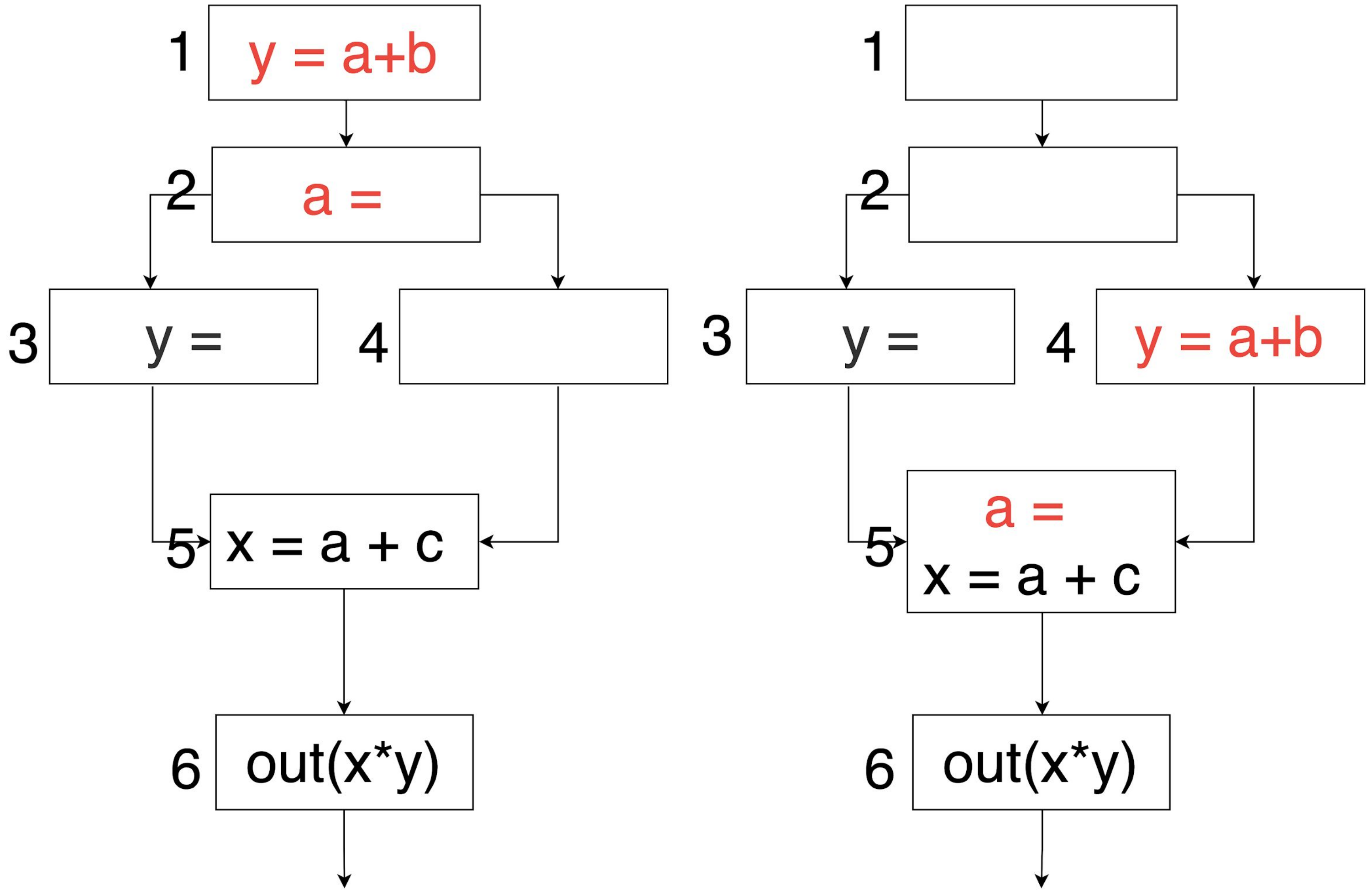
1. Move the instruction through all kinds of control flow to reach its “live” branch
2. Have to deal with “faint” code situation
3. Second order effect: eliminating partial dead code might create further elimination opportunities.



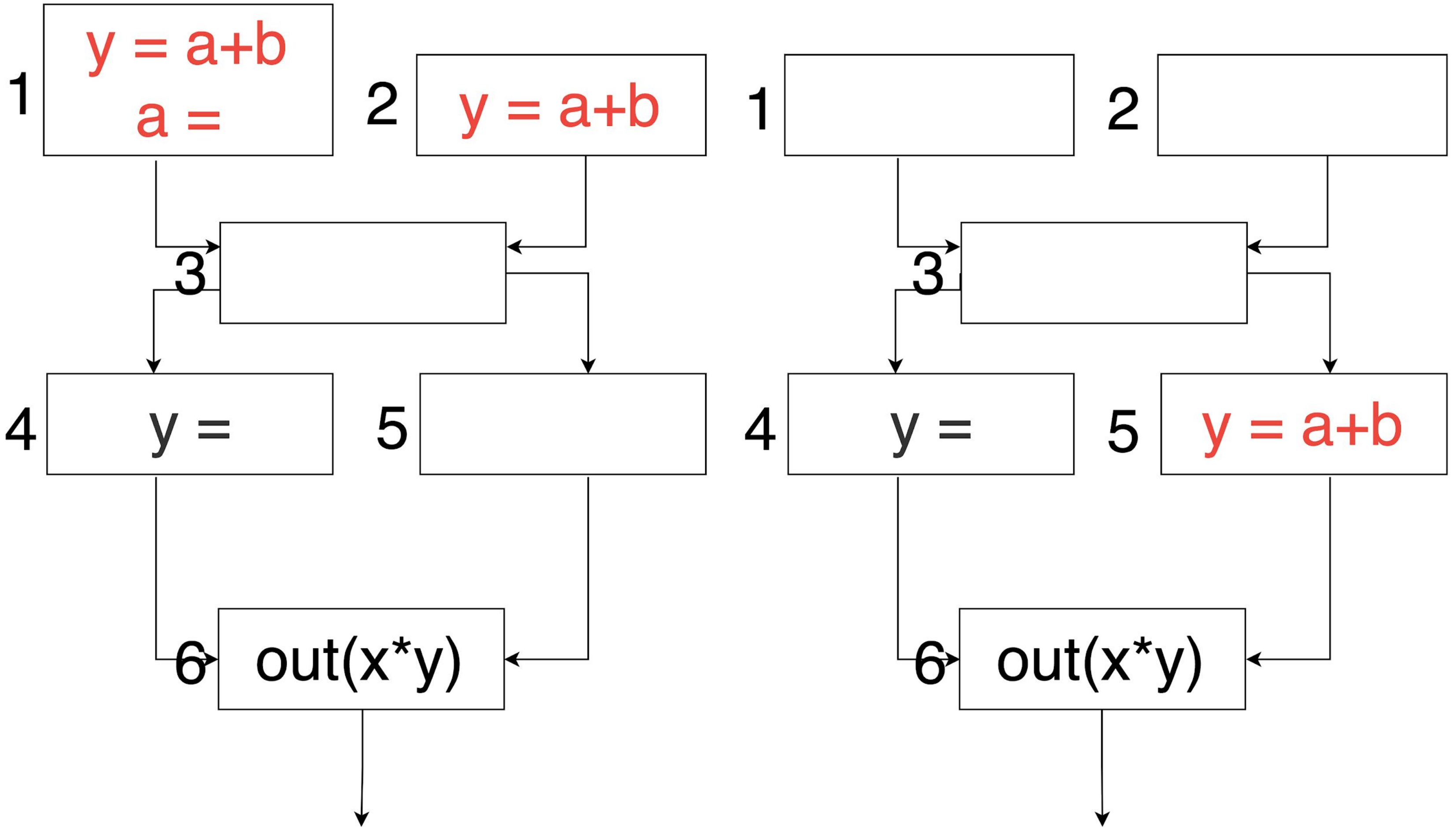
Sinking-Eliminating effects



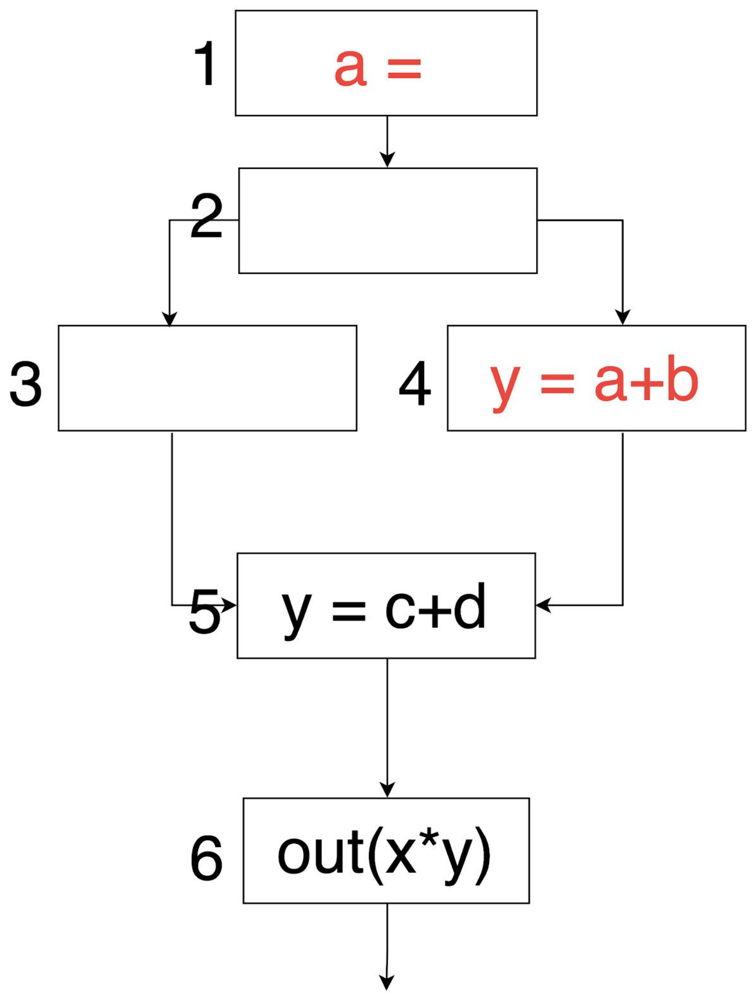
Sinking-Sinking effects

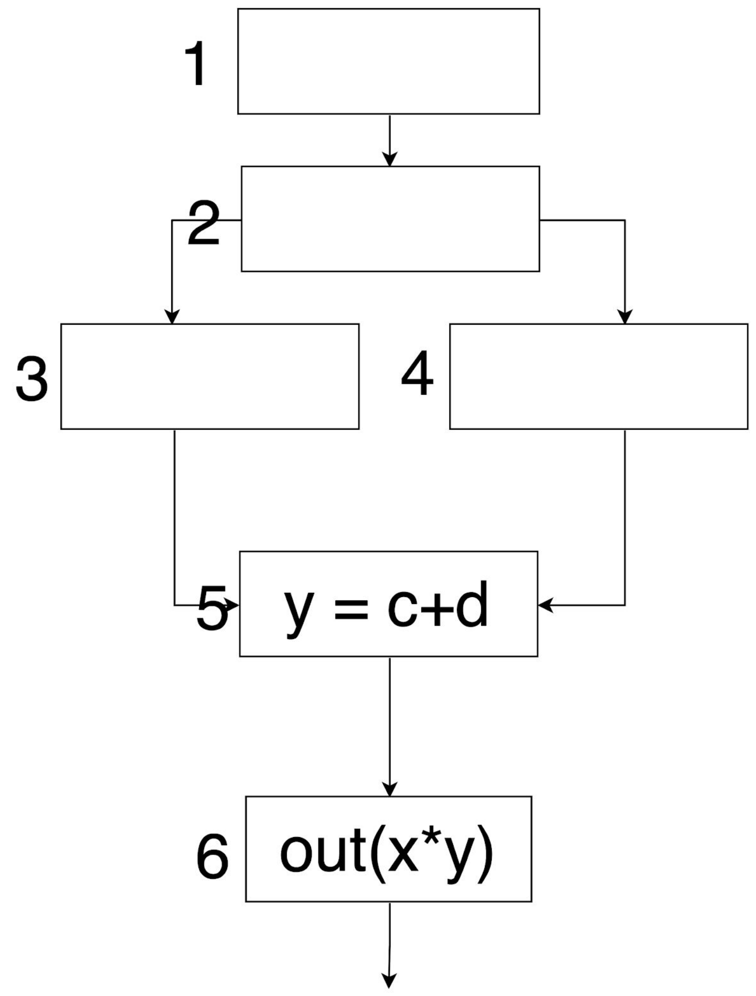


Elimination-Sinking effects



Elimination-elimination effects





The Algorithm

1. Dead(faint) variable analysis
2. Dead(faint) assignments elimination
3. Delayability analysis
4. Ask for assignment sinking
5. Repeat 1-4 until the program becomes invariant

Some predicates:

ASS-USED(l, x)

N-DELAYED(n, a), X-DELAYED(n, a), N-INSERT(n, a), X-INSERT(n, a)

LOCDELAYED(n, a), LOCBLOCKED(n, a)

The Algorithm

The Dead Variable Analysis:

Trivial using DU chain

The Faint Variable Analysis:

```
function isFaint(l)
    if isDead(l)
        return true
    else
        for u in uses of l
            if lhs(l).def == l
                continue
            elif ASS-USED(u, l) and !isFaint(lhs(l).def)
                return false
    return true
```

The Algorithm

Delayability Analysis (forward):

$$\text{N-DELAYED}(n, a) = \prod_{m \in pred(n)} \text{X-DELAYED}(m, a)$$

$$\text{X-DELAYED}(n, a) = \text{LOCDELAYED}(n, a)$$

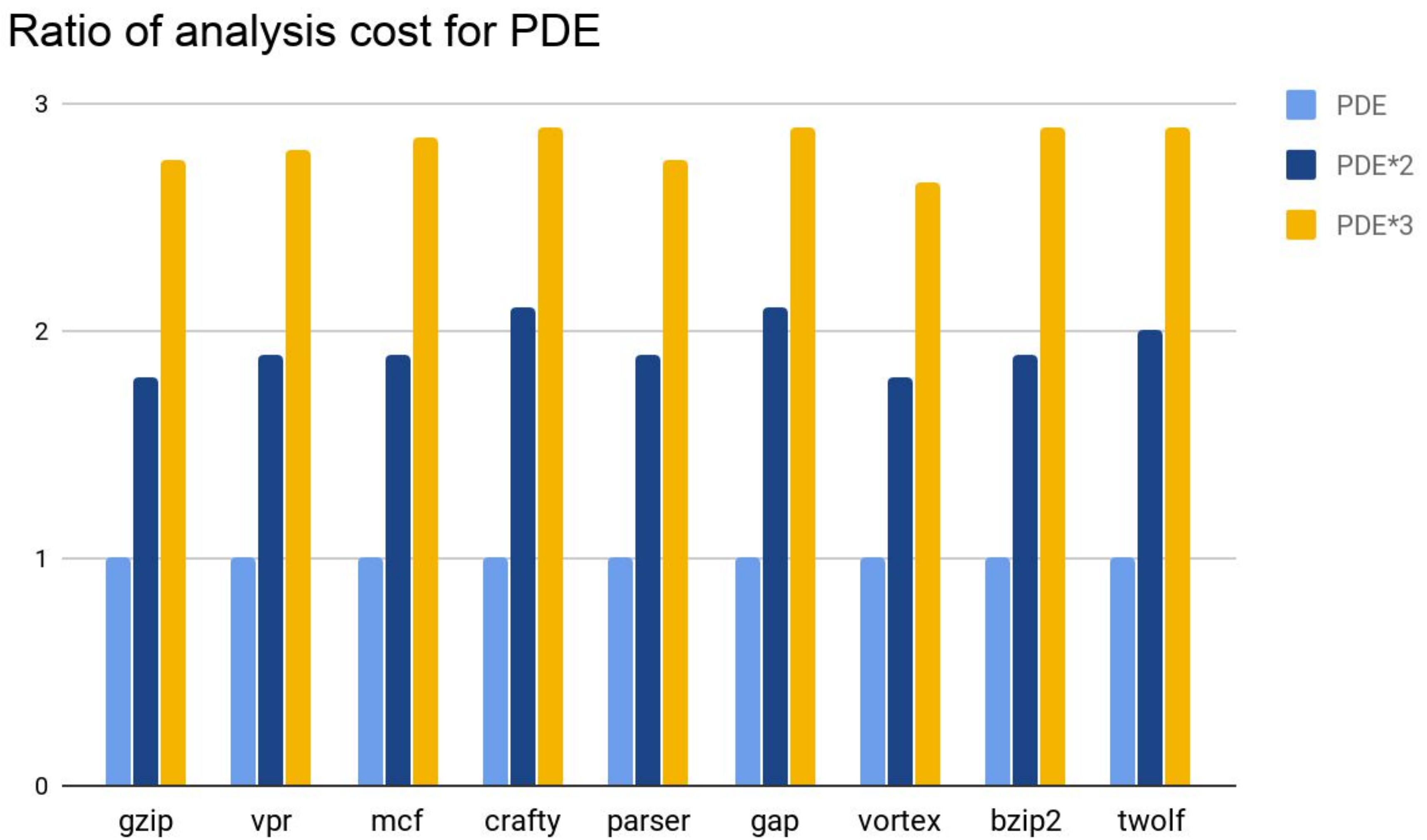
$$+ \text{ N-DELAYED}(n, a) * \neg\text{LOCBLOCKED}(n, a)$$

Insertion Points:

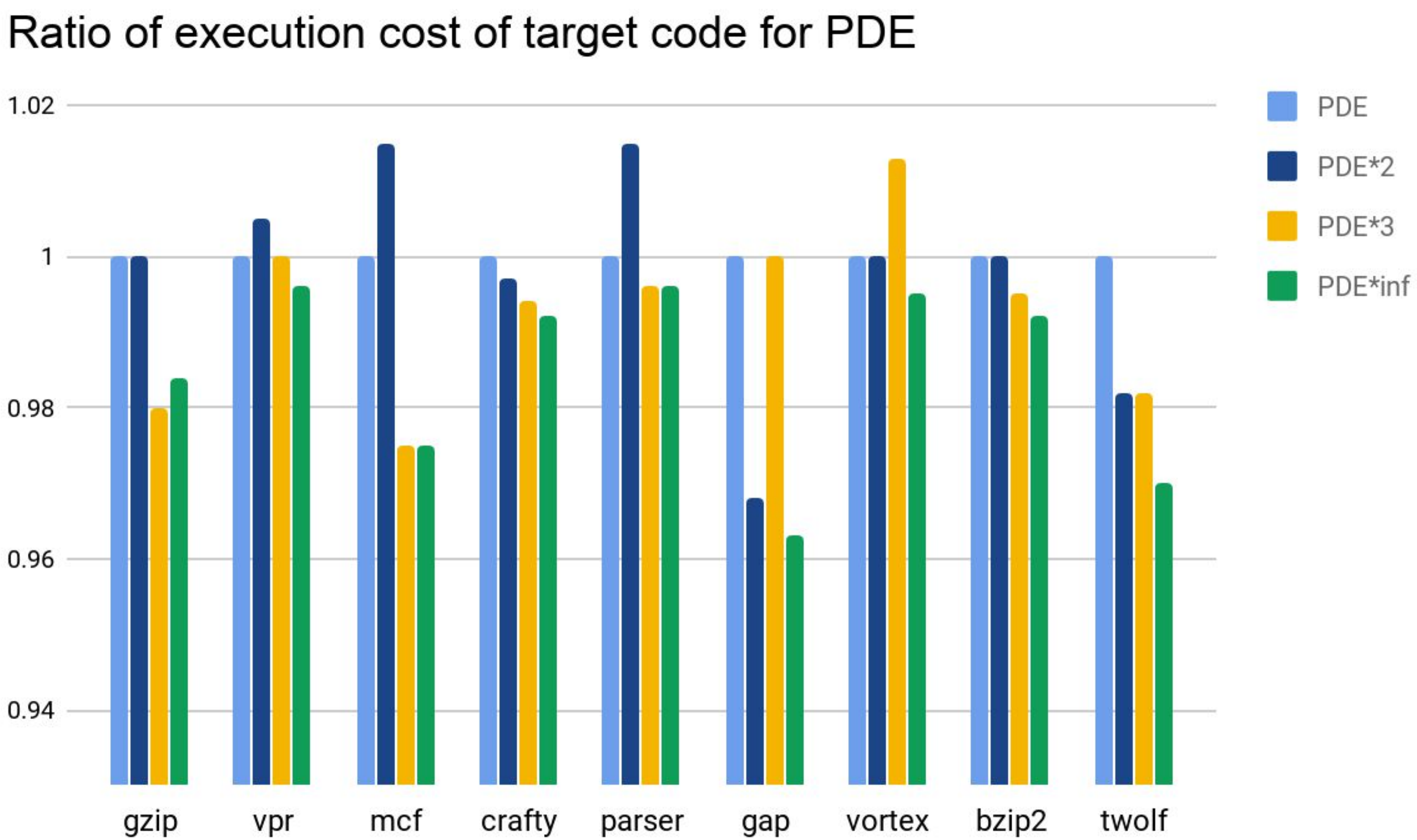
$$\text{N-INSERT}(n, a) = \text{N-DELAYED}(n, a) * \text{LOCBLOCKED}(n, a)$$

$$\text{X-INSERT}(n, a) = \text{X-DELAYED}(n, a) * \sum_{m \in succ(n)} \neg\text{N-DELAYED}(m, a)$$

The Result


Ratio of analysis cost for PDE
3
2
1
0
PDE
PDE*2
PDE*3
gzip
vpr
mcf
crafty
parser
gap
vortex
bzip2
twolf

The Result


Ratio of execution cost of target code for PDE
1.02
1
0.98
0.96
0.94
gzip
vpr
mcf
crafty
parser
gap
vortex
bzip2
twolf
PDE
PDE*2
PDE*3
PDE*inf

Discussion

- **Advantages**
  - Able to move statements out of loops or even across loops
  - Maintains original control structure
- **Disadvantages**
  - Must be applied repeatedly
  - Partial faint code elimination of order $O(n^5)$ in the worst case