BITS Pilani
Dubai Campus

COMPILER CONSTRUCTION
CS F363

Code Optimization

Dr. R. Elakkiya, AP/CS

Text Book

- ✓Aho, Lam, Sethi and Ullman, “Compilers-Principles, Techniques & Tools”, Pearson/Addison-Wesley, Second Edition, 2013.

Text Book Reading:

Chapter 6

Section 6.1 to 6.9

Code Optimization

- ✓Use algebraic identities to simplify computation
  - ✓$x + 0 = 0 + x = x$
  - ✓$x * 1 = 1 * x = x$
  - ✓$x – 0 = x$
  - ✓$x/1 = x$
  - ✓$x * 0 = 0 * x = 0$

Code Optimization

✓Reduction in strength

Expensive	Cheaper
x^2	x * x
2 * x	x + x
x / 2	x * 0.5

Code Optimization

- ✓Constant folding:
  - ✓Evaluate expression during compile time, if both operands values of an operation are constants. Replace the constant expressions by its value.
  - ✓Replace variables that have constant values with their values.
- ✓Ex. A = 2, B = 3.14, C = A * B. Can be replaced as C = 6.28
- ✓**Ex:** `tmp=5*3+8-12/2` `tmp=17`

Code Optimization

- ✓Commutativity and associativity
- ✓x * y = y * x

- ✓a = b + c;
- ✓e = c + d + b;

- ✓a = b + c
- ✓t = c + d
- ✓e = t + b

If t is not needed outside this block, we can change this sequence to
a = b + c
e = a + d

BITS Pilani
Dubai Campus

innovate achieve lead

Code Optimization

- ✓ Common sub-expressions: Instructions that compute a value that has already been computed
- ✓ Types:
  - ✓ Local common sub-expression: common sub-expressions that appear in single basic block
  - ✓ Global common sub-expression: common sub-expressions that appear across basic block
  - ✓ Examples

a = 10;
b = a + 1;
c = a + 1;
d = c + a;

a = 10;
c = a + 1;
d = c + a;
Assuming b's value is not needed anymore

Code Optimization

✓Consider the BB and the DAG

a = b + c

b = a - d

c = b + c

d = a - d

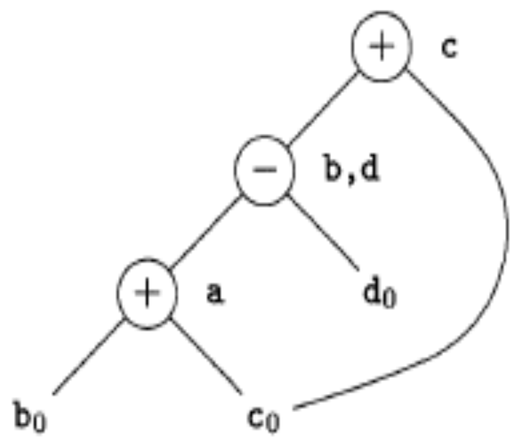
✓The optimized BB (asuming b is not live)

a = b + c

d = a - d

c = d + c

b = d (required if b is live)



Code Optimization

✓Consider the BB and the DAG

```
d = b * c
e = a + b
b = b * c
a = e - d
```

✓The optimized BB (asuming a is live)

```
d = b * c
e = a + b
a = e - d
```

Code Optimization

- ✓Dead code
  - ✓Instructions that compute a value that is never used
  - ✓Code in a branch that is never taken

Code Optimization

- ✓Copy propagation:
- ✓After a ‘copy’ statement, x=y, try to use y as far as possible

```
t=I*4                                   t=I*4
s=t                                     s=t
a=s+4                                   a=t+4
```

Code Optimization

- Code motion: Move the loop-invariant computation outside the loop
- Loop-invariant : An expression that gives the same result, independent of no of times loop is executed.

```
while (i<= limit-2)
{
/* value of limit is not changed
}
```

- limit-2 is computed every time the loop is executed
- If value of limit is not changed, then we can move limit-2 outside the loop

```
t = limit-2
while (i <= t)
```

Tutorial

✓Consider the following basic block, in which all variables are integers and ** denotes exponentiation.

✓Assume that the only variables that are live at the exit of this block are v and z. In order, apply the following optimizations to this basic block. Show the result of each transformation.

1. algebraic simplification
2. common sub-expression elimination
3. copy propagation
4. constant folding
5. dead code elimination

```
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w
```

Is the resulting program optimal? What optimizations, in what order, can you apply to optimize the result

Tutorial

Original BB
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w

1) Algebraic optimization:
a := b + c
z := a * a
x := 0
y := b + c
w := y * y
u := x + 3
v := u + w

2) Common sub-expression elimination:
a := b + c
z := a * a
x := 0
y := a
w := y * y
u := x + 3
v := u +

3) Copy propagation:
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 0 + 3
v := u + w

4) Constant folding:
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 3
v := u + w

5) Dead code elimination:
a := b + c
z := a * a
w := a * a
u := 3
v := u + w

Tutorial

Final optimized code

```
a := b + c
z := a * a
v := 3 + z
```

Thank you