Introduction to Machine-Independent Optimizations Part 1

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

Outline of the Lecture

- What is code optimization?
- Types of code optimizations
- Illustrations of code optimizations

Machine-independent Code Optimization

- Intermediate code generation process introduces many inefficiencies
  - Extra copies of variables, using variables instead of constants, repeated evaluation of expressions, etc.
- Code optimization removes such inefficiencies and improves code
- Improvement may be time, space, or power consumption
- It changes the structure of programs, sometimes of beyond recognition
  - Inlines functions, unrolls loops, eliminates some programmer-defined variables, etc.
- Code optimization consists of a bunch of heuristics and percentage of improvement depends on programs (may be zero also)

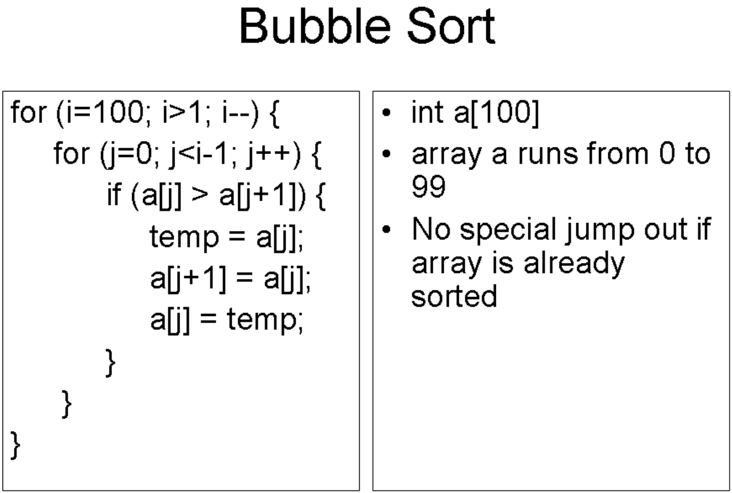
Examples of Machine-Independant Optimizations

- Global common sub-expression elimination
- Copy propagation
- Constant propagation and constant folding
- Loop invariant code motion
- Induction variable elimination and strength reduction
- Partial redundancy elimination
- Loop unrolling
- Function inlining
- Tail recursion removal
- Vectorization and Concurrentization
- Loop interchange, and loop blocking

Data-flow Analysis

- Code optimization needs information about the program
  - which expressions are being recomputed in a function?
  - Which expressions are partially redundant?
  - which definitions reach a point?
  - Which copies and constants can be propagated? Etc.
- All such information is gathered through data-flow analysis

Bubble Sort Running Example

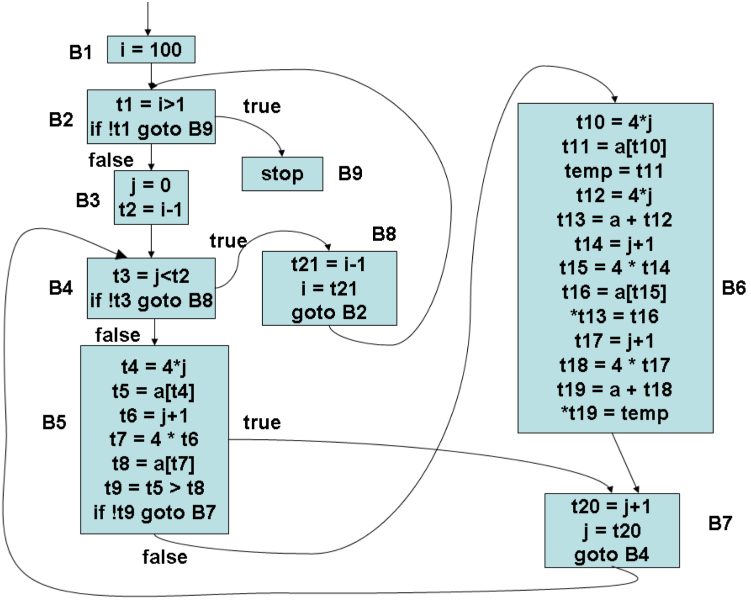


Bubble Sort

```
for (i=100; i>1; i--) {
    for (j=0; j<i-1; j++) {
        if (a[j] > a[j+1]) {
            temp = a[j];
            a[j+1] = a[j];
            a[j] = temp;
        }
    }
}
```

- int a[100]
- array a runs from 0 to 99
- No special jump out if array is already sorted

Control Flow Graph of Bubble Sort



B1
```
i = 100
```

B2
```
t1 = i>1
if !t1 goto B9
```

true

B9
```
stop
```

false

B3
```
j = 0
t2 = i-1
```

true

B8
```
t21 = i-1
i = t21
goto B2
```

B4
```
t3 = j<t2
if !t3 goto B8
```

false

B5
```
t4 = 4*j
t5 = a[t4]
t6 = j+1
t7 = 4 * t6
t8 = a[t7]
t9 = t5 > t8
if !t9 goto B7
```

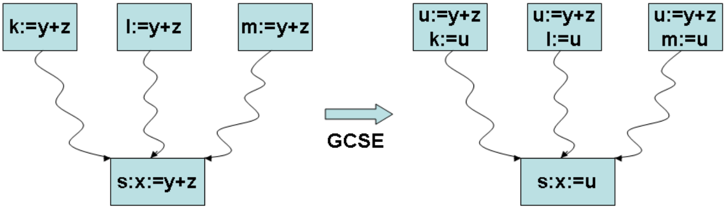
true

false

B6
```
t10 = 4*j
t11 = a[t10]
temp = t11
t12 = 4*j
t13 = a + t12
t14 = j+1
t15 = 4 * t14
t16 = a[t15]
*t13 = t16
t17 = j+1
t18 = 4 * t17
t19 = a + t18
*t19 = temp
```

B7
```
t20 = j+1
j = t20
goto B4
```

GCSE Conceptual Example



k:=y+z

l:=y+z

m:=y+z

s:x:=y+z

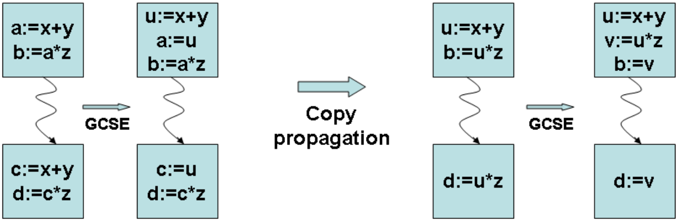
GCSE

u:=y+z
k:=u

u:=y+z
l:=u

u:=y+z
m:=u

s:x:=u



a:=x+y
b:=a*z

c:=x+y
d:=c*z

GCSE

u:=x+y
a:=u
b:=a*z

c:=u
d:=c*z

Copy propagation

u:=x+y
b:=u*z

d:=u*z

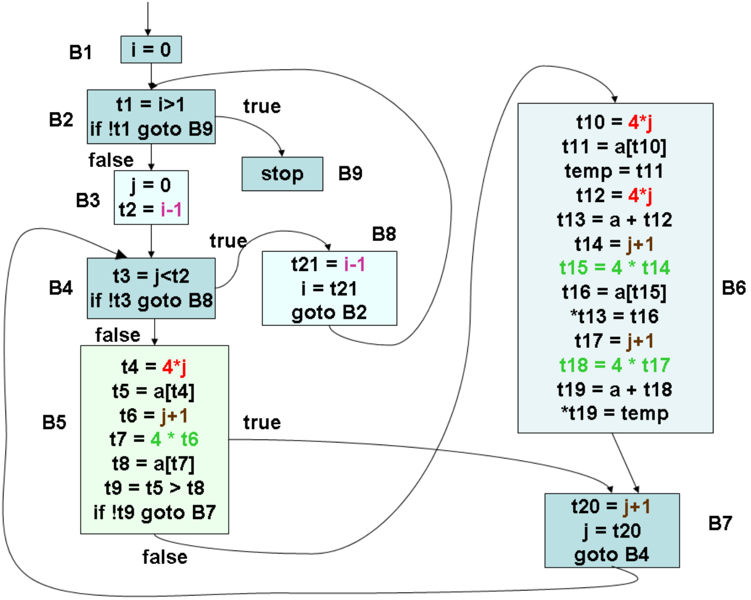
GCSE

u:=x+y
v:=u*z
b:=v

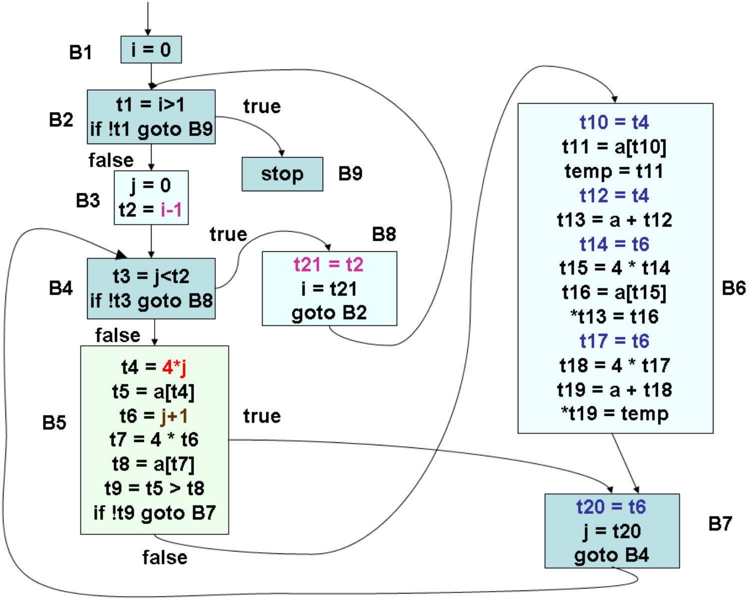
d:=v

Demonstrating the need for repeated application of GCSE

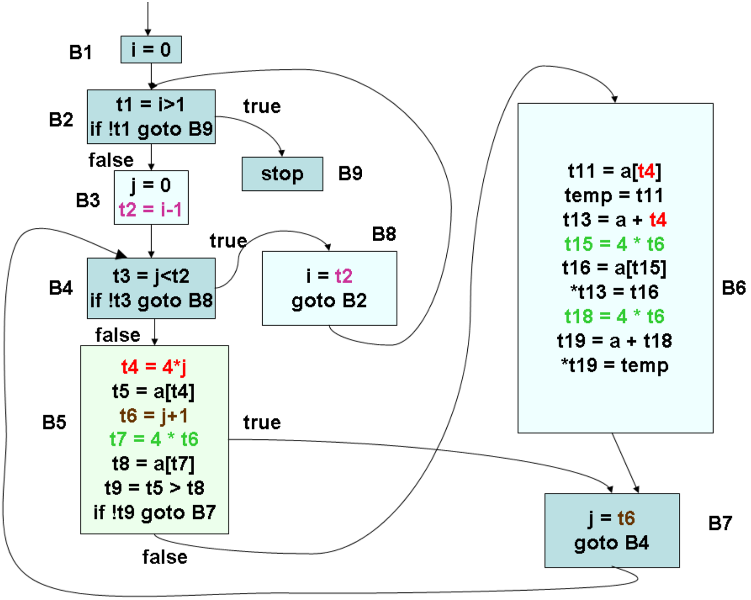
GCSE on Running Example - 1



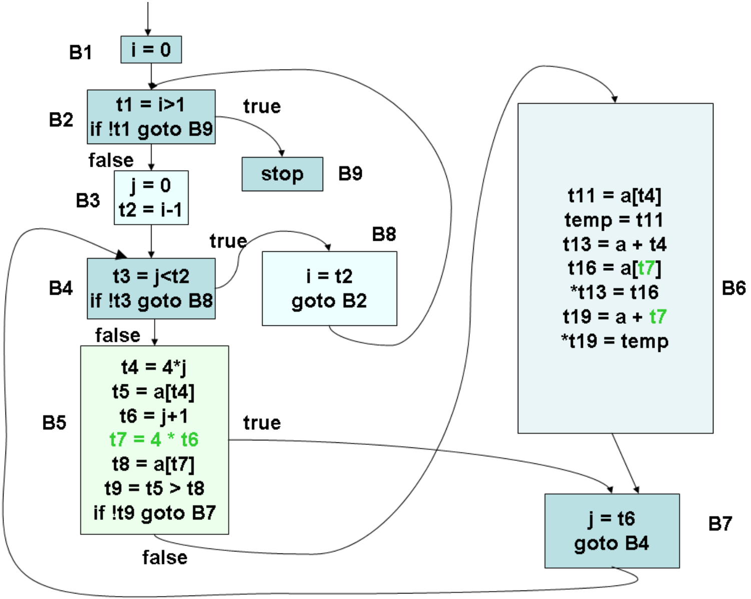
GCSE on Running Example - 2



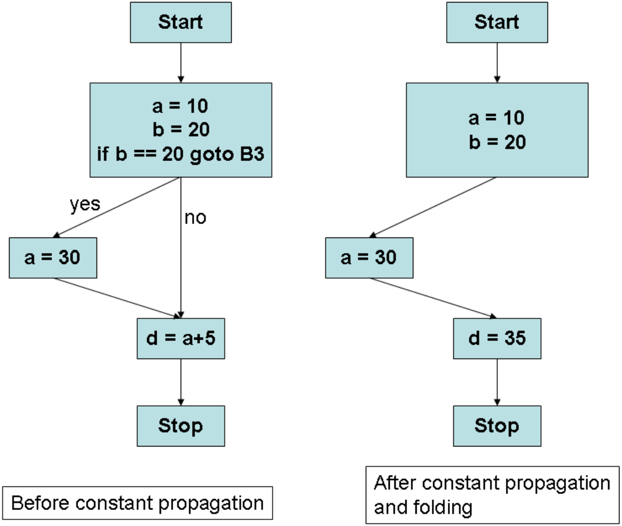
Copy Propagation on Running Example



GCSE and Copy Propagation on Running Example



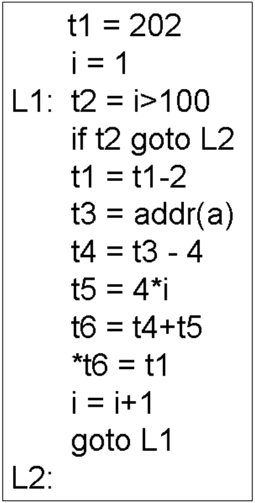
Constant Propagation and Folding Example



Before constant propagation

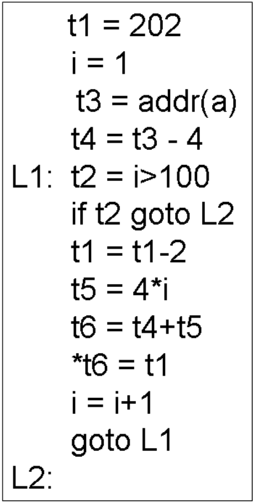
After constant propagation and folding

Loop Invariant Code motion Example



```
     t1 = 202
     i = 1
L1:  t2 = i>100
     if t2 goto L2
     t1 = t1-2
     t3 = addr(a)
     t4 = t3 - 4
     t5 = 4*i
     t6 = t4+t5
     *t6 = t1
     i = i+1
     goto L1
L2:
```

Before LIV code motion



```
     t1 = 202
     i = 1
     t3 = addr(a)
     t4 = t3 - 4
L1:  t2 = i>100
     if t2 goto L2
     t1 = t1-2
     t5 = 4*i
     t6 = t4+t5
     *t6 = t1
     i = i+1
     goto L1
L2:
```

After LIV code motion

Strength Reduction

```
      t1 = 202
      i = 1
      t3 = addr(a)
      t4 = t3 - 4
L1:   t2 = i>100
      if t2 goto L2
      t1 = t1-2
      t5 = 4*i
      t6 = t4+t5
      *t6 = t1
      i = i+1
      goto L1
L2:
```

Before strength reduction for t5

```
      t1 = 202
      i = 1
      t3 = addr(a)
      t4 = t3 – 4
      t7 = 4
L1:   t2 = i>100
      if t2 goto L2
      t1 = t1-2
      t6 = t4+t7
      *t6 = t1
      i = i+1
      t7 = t7 + 4
      goto L1
L2:
```

After strength reduction for t5 and copy propagation

Induction Variable Elimination

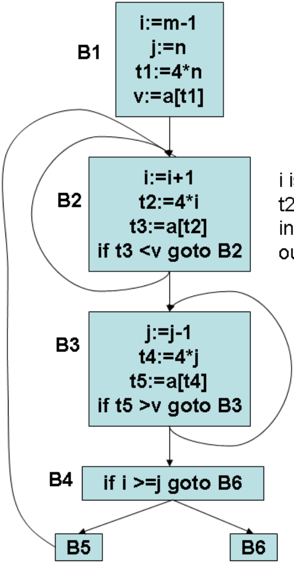
```
       t1  = 202
       i = 1
       t3 = addr(a)
       t4 = t3 – 4
       t7 = 4
L1:  t2 = i>100
       if t2 goto L2
       t1  = t1-2
       t6 = t4+t7
       *t6 = t1
       i = i+1
       t7 = t7 + 4
       goto L1
L2:
```

Before induction variable elimination (i)

```
       t1  = 202
       t3 = addr(a)
       t4 = t3 – 4
       t7 = 4
L1:  t2 = t7 >400
       if t2 goto L2
       t1  = t1-2
       t6 = t4+t7
       *t6 = t1
       t7 = t7 + 4
       goto L1
L2:
```

After eliminating i and replacing it with t7

Induction Variable Elimination and Strength Reduction



B1

i:=m-1
j:=n
t1:=4*n
v:=a[t1]

B2

i:=i+1
t2:=4*i
t3:=a[t2]
if t3 <v goto B2

B3

j:=j-1
t4:=4*j
t5:=a[t4]
if t5 >v goto B3

B4

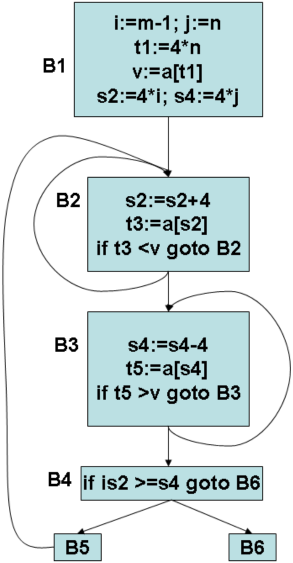
if i >=j goto B6

B5

B6

i is basic i.v
t2 is derived i.v
in both inner and
outer loops

j is basic i.v
t4 is derived i.v
in both inner
and outer loops



B1

i:=m-1; j:=n
t1:=4*n
v:=a[t1]
s2:=4*i; s4:=4*j

B2

s2:=s2+4
t3:=a[s2]
if t3 <v goto B2

B3

s4:=s4-4
t5:=a[s4]
if t5 >v goto B3

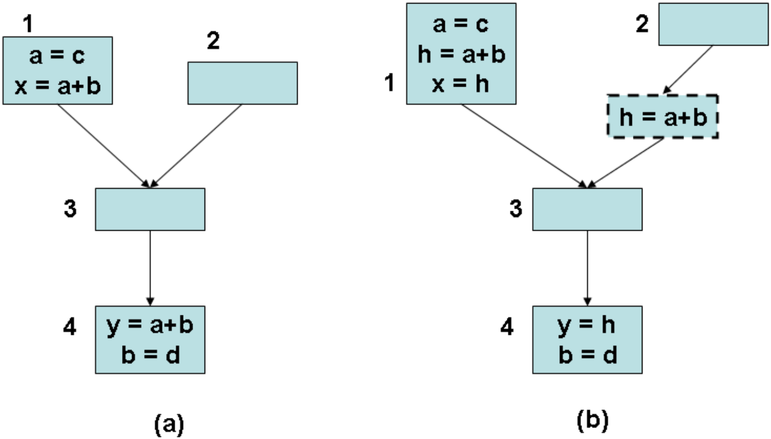
B4

if is2 >=s4 goto B6

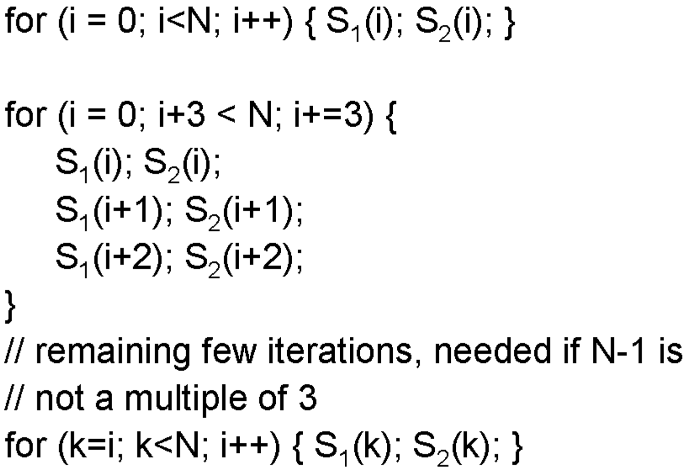
B5

B6

Partial Redundancy Elimination



Unrolling a For-loop



```
for (i = 0; i<N; i++) { S_1(i); S_2(i); }

for (i = 0; i+3 < N; i+=3) {
    S_1(i); S_2(i);
    S_1(i+1); S_2(i+1);
    S_1(i+2); S_2(i+2);
}
// remaining few iterations, needed if N-1 is
// not a multiple of 3
for (k=i; k<N; i++) { S_1(k); S_2(k); }
```