Compiler Design

Lecture-23

Introduction to Code Motion

Topics Covered

- Code Motion
- Dead Code Elimination

Code Motion

- Moving code from one part of the program to other without modifying the algorithm
  - Reduce size of the program
  - Reduce execution frequency of the code subjected to movement

Code Motion

1. *Code Space reduction*: Similar to common sub-expression elimination but with the objective to reduce code size.

Example: Code hoisting

```
if (a< b) then
    z := x ** 2
else
    y := x ** 2 + 10
```

```
temp : = x ** 2
if (a< b) then
   z := temp
else
  y := temp + 10
```

"x ** 2" is computed once in both cases, but the code size in the second case reduces.

Code Motion

2. *Execution frequency reduction*: reduce execution frequency of partially available expressions (expressions available atleast in one path)

Example:

```
if (a<b) then                    if (a<b) then
      z = x * 2                     temp = x * 2
                        ⟶           z = temp
else                             else
y = 10                             y = 10
                                   temp = x * 2
g = x * 2                                g = temp;
```

Code Motion

- Move expression out of a loop if the evaluation does not change inside the loop.

Example:

```
while ( i < (max-2) ) …
```

Equivalent to:

```
t :=  max - 2
while ( i < t ) …
```

Code Motion

- Safety of Code movement

  Movement of an expression *e* from a basic block b_i to another block b_j, is safe if it does not introduce any new occurrence of *e* along any path.

Example: Unsafe code movement

```
if (a<b) then                    temp = x * 2
   z = x * 2          ⟶         if (a<b) then
else                                z = temp
  y = 10                         else
                                 y = 10
```

Strength Reduction

- Replacement of an operator with a less costly one.

Example:

```
for i=1 to 10 do
  ...
  x = i * 5
  ...

end
```

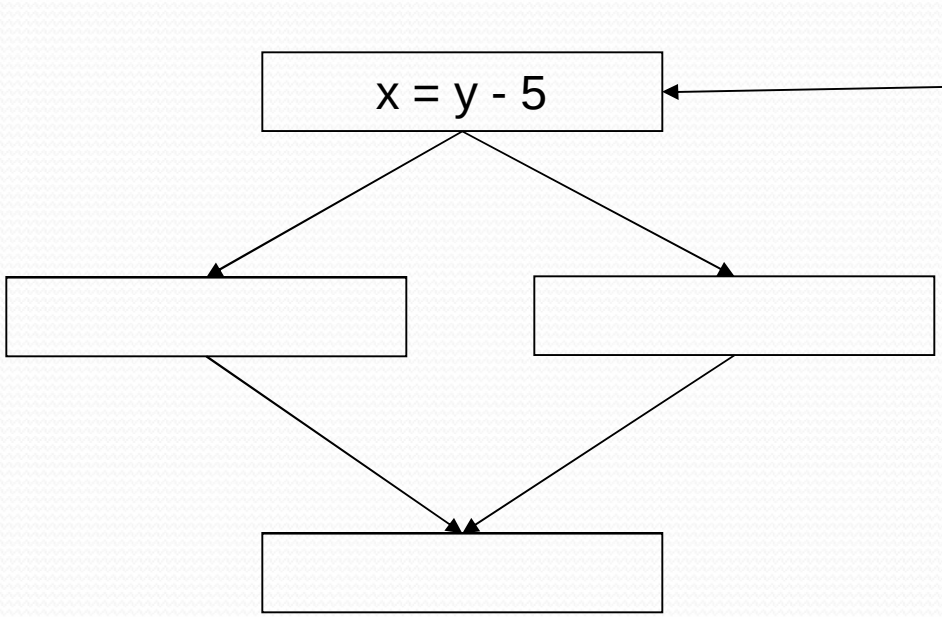
```
temp = 5;
for i=1 to 10 do
  ...
  x = temp
  ...
  temp = temp + 5
end
```

- Typical cases of strength reduction occurs in address calculation of array references.
- Applies to integer expressions involving induction variables (loop optimization)

Dead Code Elimination

- Dead Code are portion of the program which will not be executed in any path of the program.
  - Can be removed
- Examples:
  - No control flows into a basic block
  - A variable is dead at a point -> its value is not used anywhere in the program
  - An assignment is dead -> assignment assigns a value to a dead variable

Dead Code Elimination



"x" is dead variable

Definition of "x" is dead

- Beware of side effects in code during dead code elimination

Dead Code Elimination

- Examples:

```
DEBUG:=0
if (DEBUG) print
```

← Can be eliminated

Copy Propagation

- What does it mean?
  - Given an assignment x = y, replace later uses of x with uses of y, provided there are no intervening assignments to x or y.
- When is it performed?
  - At any level, but usually early in the optimization process.
- What is the result?
  - Smaller code

Copy Propagation

- f := g are called copy statements or copies
- Use of g for f, whenever possible after copy statement

Example:

x[i] = a;	x[i] = a;
sum = x[i] + a;	sum = a + a;

- May not appear to be code improvement, but opens up scope for other optimizations.

Local Copy Propagation

- Local copy propagation
  - Performed within basic blocks
  - Algorithm sketch:
    - traverse BB from top to bottom
    - maintain table of copies encountered so far
    - modify applicable instructions as you go

Loop Optimization

- Decrease the number if instruction in the inner loop
- Even if we increase no of instructions in the outer loop
- Techniques:
  - Code motion
  - Induction variable elimination
  - Strength reduction

Peephole Optimization

- Pass over generated code to examine a few instructions, typically 2 to 4
  - Redundant instruction Elimination: Use algebraic identities
  - Flow of control optimization: removal of redundant jumps
  - Use of machine idioms

Redundant instruction elimination

- Redundant load/store: see if an obvious replacement is possible

```
MOV  R0, a
MOV a, R0
```

  Can eliminate the second instruction without needing any global knowledge of ***a***

- Unreachable code: identify code which will never be executed:

```
#define DEBUG 0
if( DEBUG) {
    print debugging info
}
```

→

```
if (0 != 1) goto L2
print debugging info


L2:
```

Algebraic identities

- Worth recognizing single instructions with a constant operand:

```
A * 1 = A
A * 0 = 0
A / 1 = A
  A * 2 = A + A
```

  More delicate with floating-point

- Strength reduction:

```
A ^ 2 = A * A
```