Program Analysis

https://www.cse.iitb.ac.in/~karkare/cs618/

Code Optimizations

Amey Karkare
Dept of Computer Science and Engg
IIT Kanpur
Visiting IIT Bombay
karkare@cse.iitk.ac.in
karkare@cse.iitb.ac.in

Recap

- Optimizations
  - To improve efficiency of generated executable (time, space, resources …)
  - Maintain semantic equivalence
- Two levels
  - Machine Independent
  - Machine Dependent

Peephole Optimization

- target code often contains redundant instructions and suboptimal constructs
- examine a short sequence of target instruction (peephole) and replace by a shorter or faster sequence
- peephole is a small moving window on the target systems

Peephole optimization examples…

Redundant loads and stores

- Consider the code sequence

  Move R_0, a
  Move a, R_0

- Instruction 2 can always be removed if it does not have a label.

Peephole optimization examples…

Unreachable code

- Consider following code sequence

```
int  debug = 0
if (debug) {
        print debugging info
}
```

this may be translated as

```
        if debug == 1 goto L1
        goto L2
L1: print debugging info
L2:
```

Eliminate jumps

```
        if debug != 1 goto L2
                print debugging information
L2:
```

5

Unreachable code example …

constant propagation

```
        if 0 <> 1 goto L2
           print debugging information
    L2:
```

Evaluate boolean expression. Since if condition is always true the code becomes

```
        goto L2
        print debugging information
    L2:
```

The print statement is now unreachable. Therefore, the code becomes

```
    L2:
```

6

Peephole optimization examples…

- flow of control: replace jump over jumps

```
    goto L1                          goto L2
    …                    by          …
    …                                …
L1 : goto L2                     L1: goto L2
```

- Simplify algebraic expressions

    remove x := x+0 or x:=x*1

7

Peephole optimization examples…

- Strength reduction
  - Replace X^2 by X*X
  - Replace multiplication by left shift
  - Replace division by right shift
- Use faster machine instructions

```
replace    Add #1,R
by         Inc R
```

8

Course Logistics

Proposed Evaluation

Assignments	**5%-10%**
Course Project	**30%-40%**
(Proposal	**5%)**
(Report	**15%)**
(Implementation & Presentation	**15%)**
Mid semester exam	**10%-20%**
End semester exam	**25%-35%**
Quizzes/Class Participation	**5%**

Audit: Stuff in BLACK

Machine Independent Optimizations

- Scope of optimizations
  - Local
  - Global
  - Interprocedural

Intraprocedural

Local Optimizations

- Restricted to a basic block
- Simplifies the analysis
- Not all optimizations can be applied locally
  - E.g. Loop optimizations
- Gains are also limited
- Simplify global/interprocedural optimizations

13

Global Optimizations

- Typically restricted within a procedure/function
  - Could be restricted to a smaller scope, e.g. a loop
- Most compiler implement up to global optimizations
  - Well founded theory
  - Practical gains

14

Interprocedural Optimizations

- Spans multiple procedures, files
  - In some cases multiple languages!
- Not as popular as global optimizations
  - No single theory applicable to multiple scenarios
  - Time consuming

15

A Catalogue of Code Optimizations

16

Compile-time Evaluation

- Move run-time actions to compile-time
- Constant Folding:

```
Volume = 4/3*PI*r*r*r;
```

  - Compute 4/3*PI at compile time
  - Applied very frequently for linearizing indices of multidimensional arrays
  - How/When can we apply it?

17

Compile-time Evaluation

- Constant Propagation
  - Replace a variable by its "constant" value

```
i = 5;
…
j = i*4;
…
```

Replaced by

```
i = 5;
…
j = 5*4;
…
```

  - May result in application of constant folding
  - How/When can we apply it?

18

Common Subexpression Elimination

- Reuse a computation if already "available"

```
x = u+v;
…
y = u+v+w;
…
```

Replaced by

```
t0 = u+v;
x = t0;
…
y = t0+w;
…
```

- How to do it?
- When can we do it?

19

Copy Propagation

- Replace a variable by another
  - If they are guaranteed to have same value

```
i = k;
…
j = i*4;
…
```

Replaced by

```
i = k;
…
j = k*4;
…
```

  - May result in dead code, common subexpr, …
  - How to apply it?
  - When can we apply it?

20

Code Movement

- Move the code in a program
- Benefits:
  - Code size reduction
  - Reduction in the frequency of execution
- Allowed only if the meaning of the program does not change.
  - May result in dead code, common subexpr, …
  - How/When can we apply it?

21

Code Movement

- Code size reduction

  Suppose op generates a large number of machine instructions

```
if (a < b)
  u = x op y;
else
  v = x op y;
```

Replaced by

```
t1 = x op y;
if (a < b)
  u = t1;
else
  v = t1;
```

22

Code Movement

- Execution frequency reduction

```
if (a < b)
  u = …;
else
  v = x*y;
w = x*y;
```

Replaced by

```
if (a < b) {
  u  = …;
  t2 = x*y;
} else {
  t2 = x*y;
  v  = t2;
}
w = t2;
```

- How/When can we do it?

23

Loop Invariant Code Movement

- Execution frequency reduction

```
for (…) {
   …
  u = a+b;
   …
}
```

Replaced by

```
t3 = a+b;
for (…) {
   …
  u  = t3;
   …
}
```

- How/When can we do it?

24

Code Movement

- Safety of code motion
- Profitability of code motion

25

Other optimizations

- Dead code elimination
  - Remove unreachable, unused code.
  - Can we always do it?
- Strength reduction
  - Use of *low strength* operators in place of *high strength* operators.
    - i*i instead of i^2, pow(i,2)
    - i<<1 instead of i*2
  - Typically performed for integers only (Why?)

26

Data Flow Analysis

- Class of techniques to derive information about flow of data
  - along program execution paths
- Used to answer questions such as:
  - whether two identical expressions evaluate to same value
    - used in common subexpression elimination
  - whether the result of an assignment is used later
    - used by dead code elimination

27

Data Flow Abstraction

- Flow graph
  - Graph representation of paths that program may exercise during execution
  - Typically one graph per procedure
  - Graphs for separate procedure have to be combined/connected for interprocedural analysis
    - Later!
    - Single procedure, single flow graph for now.

28

Data Flow Abstraction

- Basic Blocks (bb)
- Input state/Output state for Stmt
  - Program point before/after a stmt
  - Denoted IN[s] and OUT[s]
  - Within a basic block:
    - Program point *after a stmt* is same as the program point *before the next stmt*

29