CS738: Advanced Compiler Optimizations

Overview of Optimizations

Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

Recap

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

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

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

Machine Independent Code Optimizations

Machine Independent Optimizations

- Scope of optimizations
  - Intraprocedural
    - Local
    - Global
  - Interprocedural

Interprocedural Optimizations

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

A Catalog of Code Optimizations

Compile-time Evaluation

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

$$\text{Volume} = \frac{4}{3} \times \pi \times r \times r \times r$$

  - Compute $\frac{4}{3} \times \pi$ at compile-time
  - Applied frequently for linearizing indices of multidimensional arrays
- When should we NOT apply it?

Compile-time Evaluation

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

```
i = 5          i = 5
:      can be replaced by   :
j = i * 4      j = 5 * 4
```

  - May result in the application of constant folding
- When should we NOT apply it?

Code Movement

- Move the code around in a program
- Benefits
  - Code size reduction
  - Reduction in the frequency of execution
- How to find out which code to move?

Code Movement

- Code size reduction
  - Suppose the operator $\oplus$ results in the generation of a large number of machine instructions. Then,

```
if (a < b)              t = x⊕y
   u = x⊕y      can be replaced by   if (a < b)
else                       u = t
   v = x⊕y              else
                           v = t
```

- When should we NOT apply it?

Common Subexpression Elimination

- Reuse a computation if already "available"

```
x = u + v          t = u + v
:         can be replaced by   x = t
y = u + v          :
                   y = t
```

- How to check if an expression is already available?
- When should we NOT apply it?

Copy Propagation

- Replace (use of) a variable by another variable
  - If they are guaranteed to have the "same value"

```
i = k          i = k
:      can be replaced by   :
j = i * 4      j = k * 4
```

  - May result in dead code, common subexpression
- When should we NOT apply it?

Code Movement

- Execution frequency reduction

```
if (a < b)              if (a < b)
   u = . . .               u = . . .
else                       t = x * y
   v = x * y    can be replaced by   else
w = x * y                  t = x * y
                           v = t
                        w = t
```

- When should we NOT apply it?

Loop Invariant Code Movement

- Move loop invariant code out of the loop

```
for (. . . ) {              t = a + b
   . . .                    for (. . . ) {
   u = a + b    can be replaced by     . . .
   . . .                       u = t
}                              . . .
                            }
```

- When should we NOT apply it?

Code Movement

Safety of code motion

Profitability of code motion

Other Optimizations

- Dead code elimination
  - Remove unreachable and/or unused code.
  - Can we always do it?
  - Is there ever a need to introduce unused code?
- Strength Reduction
  - Use of *low strength* operators in place of *high* strength ones.
    - $i * i$ instead of $i * * 2$, pow(i, 2)
    - $i << 1$ instead of $i * 2$
  - Typically performed for integers only – Why?

3-address Code Format

- Assignments
  - x = y op z
  - x = op y
  - x = y
- Jump/control transfer
  - goto L
  - if x relop y goto L
- Statements can have label(s)
  - L: ...
- Arrays, Pointers and Functions to be added later when needed

Agenda

- Static analysis and compile-time optimizations
- For the next few lectures
- *Intraprocedural* Data Flow Analysis
  - Classical Examples
  - Components

Assumptions

- Intraprocedural: Restricted to a single function
- Input in 3-address format
- Unless otherwise specified