#### 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

### Recap

- Optimizations
  - To improve efficiency of generated executable (time, space, resources ...)
  - Maintain semantic equivalence

### Recap

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

• Scope of optimizations

• Scope of optimizations

– Local

- Scope of optimizations
  - Local
  - Global

- Scope of optimizations
  - Local
    Global
    Intraprocedural

- Scope of optimizations
  - Local
     Global
     Intraprocedural
  - Interprocedural

# 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**

# **Global Optimizations**

- Typically restricted within a procedure/function
  - Could be restricted to a smaller scope, e.g. a loop

# **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

• Spans multiple procedures, files

– In some cases multiple languages!

Spans multiple procedures, files

– In some cases multiple languages!

• Not as popular as global 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

A Catalogue of Code Optimizations

• Move run-time actions to compile-time

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

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

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

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

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

- Compute 4/3\*PI at compile time

- 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

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

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

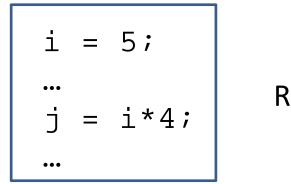
- Compute 4/3\*PI at compile time
- Applied very frequently for linearizing indices of multidimensional arrays
- When can we apply it?

• Constant Propagation

• Constant Propagation

- Replace a variable by its "constant" value

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



Replaced by

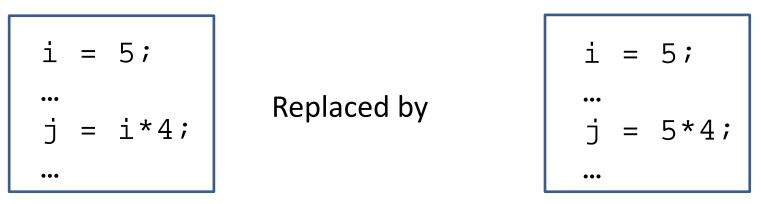
i	=	5;
 j	=	5*4;
•••		

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



May result in application of constant folding

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



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

• Reuse a computation if already "available"

• Reuse a computation if already "available"

Replaced by

• Reuse a computation if already "available"

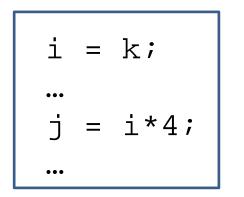
Replaced by

• When can we do it?

• Replace a variable by another

If they are guaranteed to have same value

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



Replaced by

i	=	k;
 j 	=	k*4;

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



- May result in dead code, common subexpr, ...

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



- May result in dead code, common subexpr, ...
- When can we apply it?

• Move the code in a program

- Move the code in a program
- Benefits:
  - Code size reduction
  - Reduction in the frequency of execution

- 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.

- 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, ...

- 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, ...
  - When can we apply it?



• Code size reduction



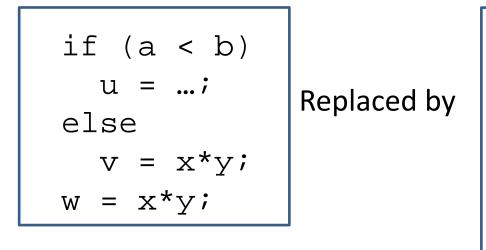
• Code size reduction

Suppose op generates a large number of machine instructions

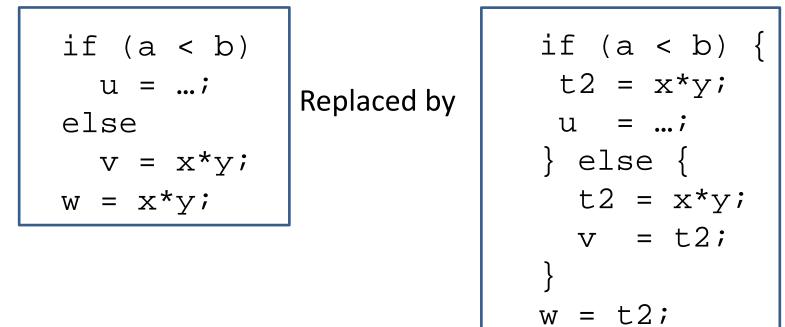
if (a < b)
 u = x op y;
else
 v = x op y;
</pre>
Replaced by
t1 = x op y;
u = t1;
else
 v = t1;

• Execution frequency reduction

Execution frequency reduction



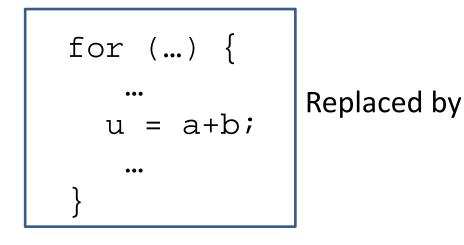
Execution frequency reduction



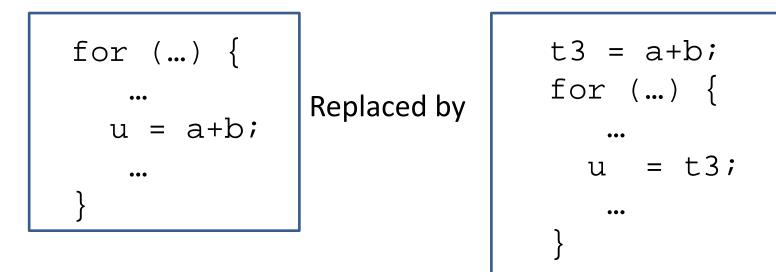
• When can we do it?

• Execution frequency reduction

• Execution frequency reduction



Execution frequency reduction



• When can we do it?

# Code MovementSafety of code motion

- Safety of code motion
- Profitability of code motion

#### Other optimizations

# Other optimizations

- Dead code elimination
  - Remove unreachable, unused code.
  - Can we always do it?

# 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?)

 Class of techniques to derive information about flow of data

along program execution paths

 Class of techniques to derive information about flow of data

along program execution paths

• Used to answer questions such as:

- 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

- 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

- 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.

• Basic Blocks (bb)

- Basic Blocks (bb)
- Input state/Output state for Stmt
  - –Program point before/after a stmt
  - –Denoted IN[s] and OUT[s]

- 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