UNIT 5

CODE OPTIMIZATION

INTRODUCTION TO CODE OPTIMIZATION -

1. Organization of the code optimizer -

   FRONT END   CODE OPTIMIZER   CODE GENERATOR

   CONTROL FLOW ANALYSIS   DATA FLOW ANALYSIS   TRANSFORMATIONS

2. Principal Source of Optimization -
   A transformation of a program is called global if it can be performed by
   looking only at the statement in the basic block otherwise it is called global.

Function-Processing Transformation -

   Various types of transformation includes -

   (i) Common Subexpression Elimination - An occurrence of an expression E is
       called a common subexpression if E was previously computed, and the values of
       variables in E have not changed since the previous computation.

   (ii) Copy Propagation - In this transformation is to use g for f, whenever
       possible after the copy statement f := g.

   (iii) Dead Code Elimination - Remove dead or useless code, statements that
       compute values that never get used.

   (iv) Constant Folding - Producing at compile time that the value of an expression
       is constant and using the constant instead.

Hook Optimization -

   Three techniques are -

   (i) Code Motion - Decrease the amount of code in a loop, by moving code outside
       eq - while (i = limit - 2) \[ t = \text{limit} - 2; \text{while} (i = t) \]

   (ii) Induction Variable Elimination - A variable is incremented/decremented by
       some constant every time then it is called induction variable. If there or more
       induction variables in a loop, it may be possible to get rid of all but one.

   (iii) Reduction in Strength - It replaces an expensive operation by cheaper one such
       eq - as a multiplication by an addition.
       \[ qg \text{ for } (\text{int } i = 10; i < 10; i++) \] \[ \Rightarrow \text{for } (\text{int } i = 4; i < 10; i++) \]

       \[ \text{count } = i + 7; \] \[ \text{count } = \text{count } + 7 + \text{count } \]

my companion
3. **Optimization of basic block** - Sometimes by constructing a DAG from a basic block
   (i) **Structural Transformations** - Various transformations on -
      (1) Common subexpression elimination
      (2) Dead code elimination
   (iii) **Algebraic Transformations** -
      (1) Reduction in strength \( g - n \times x \rightarrow n^2 \)
      (2) Use of algebraic identities \( g - n + 0 = 0 + n \rightarrow n, n \times 1 = 1 \times n \rightarrow n \)
      (3) Simple algebraic transformation
   (4) Constant folding \( g - 2 \times 3.14 \rightarrow 6.28 \)

4. **Loops in Flow Graphs** -
   (1) **Dominators** -

   In a flow graph, a node \( d \) dominates \( n \) if every path to node \( n \) from initial node goes through \( d \) only. This can be denoted as \( \text{d dom n} \). Every initial node dominates all the remaining nodes in the flow graph. Similarly, every node dominates itself.

   ![Diagram of flow graph showing dominator tree]

   (2) **Natural Loops** -

   The heads dominate their tails i.e. if \( a \rightarrow b \) is an edge, \( b \) is the head and \( a \) is tail. Then the edge \( b \) is called back edge.

   Given a back edge \( n \rightarrow d \), we define the natural loop of the edge to be \( d \) plus the set of nodes that can reach \( n \) without going through \( d \). Node \( d \) is the heads of the group.

   Algorithm for constructing the natural loop is given as:

   ```
   procedure insert(m);
   if m is not in loop then begin
   ```
loop := loop U \{ m \};  
push m onto stack 

end;
/* main program follows * /
stack := empty;
loop := \{ end \};
insert(n);
while stack is not empty do begin
    pop m, the first element of stack, off stack;
    for each predecessor p of m do insert(p)
end

(3) Inner loops - It is a loop that contains no loop.

4 \to 2 is an inner loop that means edge given by 2-3-4.

(4) Pre-Header - The pre-header is a new block created such that successor of
this block is the header block. All the computations that can be made before
the header block can be made before the pre-header block.

(5) Reducible flow graphs - It is flow graph in which there are two types of edges
forward edges and backward edges. These edges have following properties:
(i) The forward edge form an acyclic graph
(ii) The back edge are such edge whose head dominates their tail.

The program structure in which there is inclusive use of if-then, while-do
or goto statements generate a flow graph which is always reducible.
Introduction to Global Data Flow Analysis -

An optimizing compiler uses data flow information by a process known as data flow analysis.

\[ \text{out} [s] = \text{gen} [s] \cup (\text{in} [s] - \text{kill} [s]) \]

It works as the information at the end of a statement is either generated within the statement, or enters at the beginning and is not killed as control flows through the statement. Such equations are called data-flow equations.

Three factors on these data-flow equations are set up and solved on -

(i) Notion of generating & killing depend on the denied information.
(ii) Data-flow analysis is affected by the control construct in the program.
(iii) There are restrictions like procedure calls, points variables may vary.

Points & Paths -

Within a basic block, we talk about the point between two adjacent statements, as well as the point between before the point statement & after the last.

Reaching definitions -

A definition of a variable \( n \) is a statement that assigns a value to \( n \). These statements certainly define \( n \) called an unambiguous statement.

If these statements may define a value of \( n \) called an ambiguous statement like -

(1) A call of a procedure with \( n \) as parameter.
(2) An assignment through a pointer that could refer to \( n \).

By defining reaching definitions as we have, we sometimes allow inaccuracies. However, they are all in the "safe" or "conservative" direction.

A decision is conservative if it never leads to a change in what the program computes. In all situations of reaching definitions, it is normally conservative to assume that a definition can reach a point even if it might not.

Data Flow Analysis of Structured Programs -

Data Flow Equations for Reaching Definitions -

\[ \text{gen} [s] = \{ e \} \]

\[ \text{kill} [s] = D_k - \{ e \} \]

\[ \text{out} [s] = \text{gen} [s] \cup (\text{in} [s] - \text{kill} [s]) \]
This is related to definition of whether or not

\[ \text{gen}[s] = \text{gen}[s_2] \cup (\text{gen}[s_3] \cap \text{kill}[s_2]) \]
\[ \text{kill}[s] = \text{kill}[s_2] \cup (\text{kill}[s_3] \cap \text{gen}[s_2]) \]
\[ \text{in}[s] = \text{in}[s_2] \]
\[ \text{out}[s] = \text{out}[s_2] \]

Conservative estimation of data-flow information:

Conservative refers to making safe assumptions when insufficient information is available at compile time, i.e., the compiler has to guarantee not to change the meaning of the optimized code.

Safe refers to the fact that a reporter of reaching definition is safe.

Accuracy: The larger the reporter of reaching definition, the less information we have to apply code optimization.

Representation of sets:

Set of definitions, such as \( \text{gen}[s] \) and \( \text{kill}[s] \), can be represented compactly using bit vectors. We associate a number to each definition of interest in the flow graph. Then the bit vector representing a set of definitions will have 1 in position \( i \) if and only if the definition numbered \( i \) is in the set.

Code unrolling transformations:

Elimination of global common subexpression - considers only expansion generated by block start and not with whether it is recomputed several times within a block.

\[ t_2 : 4 \times i \]
\[ t_3 : a[t_2] \]
\[ t_4 : 4 \times i \]
\[ t_5 : a[t_4] \]
\[ t_6 : 4 \times i \]
\[ t_7 : a[t_5] \]

\[ u := 4 \times i \]
\[ t_2 = u \]
\[ t_3 = a[t_2] \]
\[ t_4 = 4 \times i \]
\[ t_5 : a[t_4] \]

\[ (15) := 4 \times i \]
\[ (18) := a[15] \]

\[ t_6 := (15) \]
\[ t_7 := (18) \]
Copy propagation elimination - statement look like \( n = y \).

Detection of loop-invariant computation -

Loop-invariant are those whose value does not change as long as code within the loop, which can be removed by performing code motion.

Elimination of induction variables -

A variable \( n \) is called induction variable of a loop \( L \) if every time the variable \( n \) changes value, it is incremented/decremented by some constant. It can be eliminated by strength reduction.

Data flow analysis of structured flow graph -

Depth first search/routing:

- Depth first spanning tree gives depth of a flow graph.
- Internal partitioning
- External graphs
- Node splitting

A region in a flow graph is a set of nodes \( N \) that includes a header, which dominates all the other nodes in a region.

Symbolic debugging of optimized code -

A symbolic debugger is a system that allows us to look at a program's data while that program is running.

The debugger is usually called when a program error occurs.

Reducing values of variables in basic blocks

Effects of global optimization

1. Induction-variable elimination
2. Global common subexpression elimination
3. Code Motion