UNIT IV

Code Optimization: The Principle Sources of Optimization, Basic Blocks, Optimization of Basic Blocks, Structure Preserving Transformations, Flow Graphs, Loop Optimization, Data-Flow Analysis, Peephole Optimization

CODE OPTIMIZATION

The code produced by the straight forward compiling algorithms can often be made to run faster or take less space, or both. This improvement is achieved by program transformations that are traditionally called optimizations. Compilers that apply code- improving transformations are called optimizing compilers.

Optimizations are classified into two categories. They are

Machine independent optimizations.

Machine independent optimizations are program transformations that improve the target code with-out taking into consideration any properties of the target machine.

Machine dependent optimizations:

Machine dependent optimizations are based on register allocation and utilization of special machine instruction sequences

The Principal Sources of Optimization (OR) Function preserving

The following are the principle sources of optimization techniques or function preserving transformations:

- **a)** Elimination of common subexpression
- **b)** Copy propagation
- **c)** Dead code elimination
- **d)** Constant folding

Elimination of common sub expression

- Common sub expressions can be either eliminated locally or globally.
- Local common sub expressions can be identified in a basic block.
- Hence first step to eliminate local common sub expressions is to construct a basic block.
- Then the common sub expressions in a basic block can be deleted by constructing a directed acyclic graph (DAG).
- For example, x = a + b * (a + b) + c + d

The following is the basic block.

```
t1 = a + b
 t2 = a + b
t3 = t1 * t2
t4 = t3 + c
t5 = t4 + d
```

```
x = t5
```

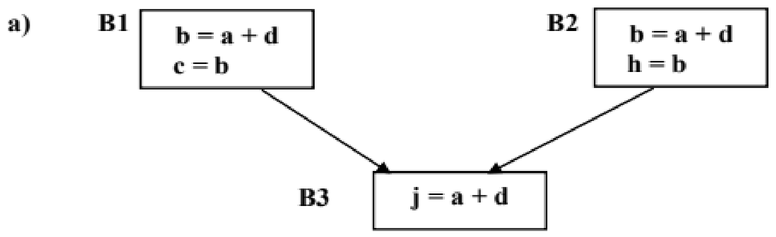
The local common sub expression in the above basic block are t1 = a + b t2 = a + b
Hence these local common sub expressions can be eliminated. The block obtained after eliminating local common sub expression is shown below.

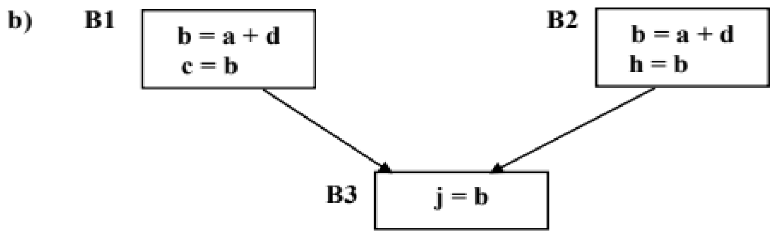
```
t1 = a + b
t2 = t1 * t1
t3 = t2 + c
t4 = t3 + d
x = t4
```

Copy or Variable propagation

- Consider the assignment statement of the form x = y. The statement x = y is called as copy statement.

To explain copy propagation, take the following example.





- Here the common sub expression is j = a + d. When a + d is eliminated, it uses j = b.

Dead code elimination

- A piece of code which is not reachable and never used anywhere in the program, then it is said to be dead code, and can be removed from the program safely.
- Generally copy statements may lead to dead code. For example, x = b + c z = x
  …
  d = x + y
- Suppose z variable is not used in the entire program, the z =x becomes the dead code. Hence, it can be optimized as,
  x = b + c
  …
  d = x + y

Constant folding

- In folding technique, the computation of a constant is done at compile time instead of execution time and further the computed value of the constant is used.
- For example, k = (22 / 7) * r * r

  Here folding is done by performing the computation of (22 / 7) at compile time. So it can be optimized as,

  ```
              k = 3.14 * r * r
  ```

Loop optimizations

- The major source of code optimization is loops, especially the inner loops.
- Most of the run time is spent inside the loops which can be reduced the number of instructions in the inner loop.
- The following are the loop optimization techniques.
  1. Code motion.
  2. Elimination of induction variables.
  3. Strength reduction.

1. Code motion:-

- Code motion is a technique which is used to move the code outside the loop.
- If there exists any expression outside the loop which the result is unchanged even after executing the loop many times, then such expression should be placed just before the loop.
- For example,

  ```
  while ( x ! = max =3 )
  {
      x = x + 2;
  }
  ```

  Here the expression max -3 is a loop invariant computation. So this can be optimized as follows:

  ```
      k = max -3;
  while ( x ! = k )
  {
      x = x + 2;
  }
  ```

2. Elimination of induction variables:-

- An induction variable is a loop control variable or any other variable that depends on the induction variable in some fixed way.
- It can also be defined as a variable which is incremented or decremented by a fixed number in the loop each time is executed.
- For example,

  ```
  for( i = 0, j = 0, k =0; i < n; i++ )
  {
      printf("%d", i );
  }
  ```

- There are three induction variables i, j, k used in the for loop. So each time i is used but j and k are not used. Hence the code can be optimized after elimination of unused induction variables is given below.

```
for( i = 0; i < n; i++ )
{
    printf("%d", i );
}
```

3. Strength reduction:-

- It is the process of replacing expensive operations by the equivalent cheaper operations on the target machine.
- For example,

```
 for( k = 1; k < 5; k++)
 {
x    = 4 * k;
 }
```

- On many machines, multiplication operation takes more time than addition or subtraction. Hence, the speed of the object code can be increased by replacing multiplication with addition or subtraction.

```
 for( k = 1; k < 5; k++)
 {
x    = x + 4;
 }
```

Optimization of Basic Blocks:

Basic block is sequence of constructive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.
Basic blocks are constructed by partitioning a sequence of three address instruction.

Algorithm for partitioning into basic blocks:

INPUT:- sequence of three address instructions

OUTPUT:- list of basic blocks

Step1

The first step is to determine the set of leaders. The rules to obtain the leaders are

1. The first statement is a leader.
2. Any statement which is the target of conditional or unconditional GOTO is a leader.
3. Any statement which immediately follows the conditional GOTO or unconditional GOTO is a leader.

Step2

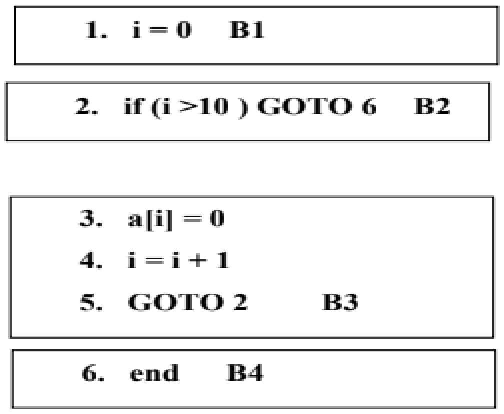
For each leader, construct the basic blocks which consists of the leader and all the instructions up to but not including the next leader or the end of the intermediate program.

- For example, let us construct the basic blocks for the following:

1. i = 0
2. if (i > 10) goto 6
3. a[i] = 0
4. i = i + 1
5. goto 2
6. end

Let us apply step1 and step2 of algorithm of algorithm to identify basic blocks.

- Based on rule 1 of step1, 1 statement is leader.
- Based on rule 2 of step1, 2 and 5 are leaders.
- Based on rule 3 of step1, 3 and 6 are leaders.

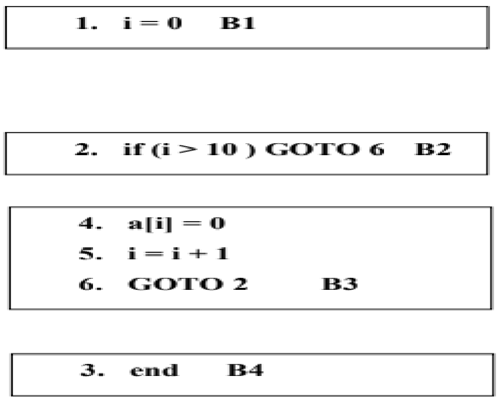


```
1. i = 0    B1

2. if (i >10 ) GOTO 6    B2

3. a[i] = 0
4. i = i + 1
5. GOTO 2        B3

6. end     B4
```

Flow graphs

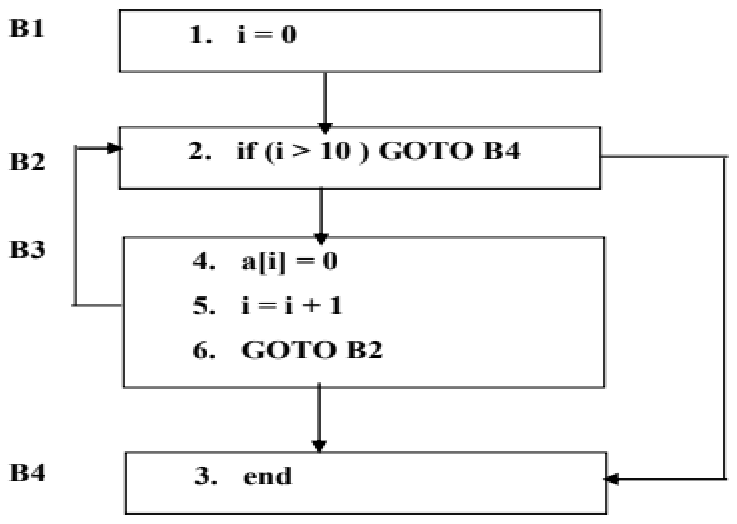
- Flow graphs are used to represent the basic blocks and their relationship by a directed graph.
- There exists an edge from block 1 to block2 if it is possible for the first instruction in block2 to immediately flow to the last instruction in block1.

For example, Let us write the flow graph for the following basic blocks.



```
1. i = 0    B1

2. if (i > 10 ) GOTO 6   B2

4. a[i] = 0
5. i = i + 1
6. GOTO 2        B3

3. end     B4
```

Flow graph for these basic blocks is:



Structure-Preserving Transformations

Optimization process can be applied on a basic block. While optimization, we don't need to change the set of expressions computed by the block.

There are two type of basic block optimization. These are as follows:

1. Structure-Preserving Transformations
2. Algebraic Transformations

1. Structure-Preserving Transformations

The primary Structure-Preserving Transformation on basic blocks is as follows:

- Common sub-expression elimination
- Dead code elimination
- Renaming of temporary variables
- Interchange of two independent adjacent statements

Common sub-expression elimination:

In the common sub-expression, you don't need to be computed it over and over again. Instead of this you can compute it once and kept in store from where it's referenced when encountered again.

```
a : = b + c
b : = a - d
c : = b + c
d : = a - d
```

In the above expression, the second and forth expression computed the same expression. So the block can be transformed as follows:

a : = b + c
b : = a - d
c : = b + c
d : = b

Dead-code elimination

- It is possible that a program contains a large amount of dead code.
- This can be caused when once declared and defined once and forget to remove them in this case they serve no purpose.
- Suppose the statement x:= y + z appears in a block and x is dead symbol that means it will never subsequently used. Then without changing the value of the basic block you can safely remove this statement.

Renaming temporary variables

A statement t:= b + c can be changed to u:= b + c where t is a temporary variable and u is a new temporary variable. All the instance of t can be replaced with the u without changing the basic block value.

Interchange of statement

Suppose a block has the following two adjacent statements:

t1 : = b + c
t2 : = x + y

These two statements can be interchanged without affecting the value of block when value of t1 does not affect the value of t2.

2. Algebraic Transformations

In the algebraic transformation, we can change the set of expression into an algebraically equivalent set. Thus the expression x:= x + 0 or x:= x *1 can be eliminated from a basic block without changing the set of expression.

Peephole Optimization

Peephole optimization is simple but effective method used to locally improve the target code by examining a short sequence of target instructions known as peephole and then replace these instructions by a shorter and/or faster sequence of instructions whenever required.
The following are peephole optimization techniques:

1. Elimination of redundant instructions.
2. Optimization of flow of control or elimination of unreachable code.
3. Algebraic simplifications.
4. Strength reduction.

Elimination of redundant instructions:-

- This includes elimination of redundant load and store instructions.
- For example,

```
MOV  R1    A
MOV  A     R1
```

- Here first instruction is storing the value of A into register R1 and second instruction is loading R1 value into A.
- These two instructions are redundant so eliminate instruction (2) because whenever instruction (2) is executed after (1), it is ensured that the register R1 contains A value.

Optimization of flow of control or elimination of unreachable code:-

- An unlabeled instruction that immediately follows an unconditional jump can be removed.
- For example, i = j if k = 2 goto L1 goto L2 L1: k is good L2:

  Here L1 immediately follows unconditional jump statement goto L2. Then the code after elimination of unreachable code is i = j if k ≠ 2 goto L2 k is good
  L2:

Algebraic simplifications:-

- Algebraic identities that occur frequently and which is worth considering them can be simplified.
- For example, X = X * 1 or X = 0 + X is often produced by straight forwards intermediate code generation algorithms. Hence they can be eliminated easily through peephole optimization.

Strength reduction:-

- Replace expensive statements by a cheaper one.

For example, X^2 is expensive operation. Hence replace this by X * X which is cheaper one.