Dr.I.Jasmine

CODE OPTIMIZATION TECHNIQUES

The code optimization in the synthesis phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources (i.e. CPU, Memory) so that faster-running machine code will result. Compiler optimizing process should meet the following objectives:

- The optimization must be correct, it must not, in any way, change the meaning of the program.
- Optimization should increase the speed and performance of the program.
- The compilation time must be kept reasonable.
- The optimization process should not delay the overall compiling process.

***There are two types of* Code Optimization Techniques**

a) **Loop Optimization is a machine independent optimization.**

This code optimization phase attempts to improve the **intermediate code** to get a better target code as the output.

b) **Peephole optimization is a machine dependent optimization technique.**

Machine-dependent optimization is done after the **target code** has been generated and when the code is transformed according to the target machine architecture.

Code Optimization is done in the following different ways:

1. Code Motion (Frequency Reduction)

In frequency reduction, the amount of code in the loop is decreased. A statement or expression, which can be moved outside the loop body without affecting the semantics of the program, is moved outside the loop.

Example:

Before optimization:

```
while(i<100)
{
        a = Sin(x)/Cos(x) + i;
       i++;
}
```

After optimization:

```
t = Sin(x)/Cos(x);
while(i<100)
{
```

```
    a = t + i;
    i++;
}
```

2. Dead Code Elimination:

- Copy propagation often leads to making assignment statements into dead code.
- A variable is said to be dead if it is never used after its last definition.
- In order to find the dead variables, a data flow analysis should be done.

Example:

Before optimization:

```
c = a * b
x = a
till
d = a * b + 4
```

After optimization:

```
c = a * b
till
d = a * b + 4
```

3. Induction Variable Elimination

If the value of any variable in any loop gets changed every time, then such a variable is known as an induction variable. With each iteration, its value either gets incremented or decremented by some constant value.

Example:

Before optimization:

```
B1
i:= i+1
x:= 3*i
y:= a[x]
if y< 15, goto B2
```

In the above example, i and x are locked, if i is incremented by 1 then x is incremented by 3. So, i and x are induction variables.

After optimization:

```
B1
i:= i+1
```

x:= x+4

y:= a[x]

if y< 15, goto B2

4. Strength Reduction

Strength reduction deals with replacing expensive operations with cheaper ones like multiplication is costlier than addition, so multiplication can be replaced by addition in the loop.

Example:

Before optimization:

```
while (x<10)
  {
        y := 3 * x+1;
        a[y] := a[y]-2;
       x := x+2;
  }
```

After optimization:

```
t= 3 * x+1;
   while (x<10)
     {
        y=t;
        a[y]= a[y]-2;
        x=x+2;
        t=t+6;
     }
```

5. Loop Invariant Method

In the loop invariant method, the expression with computation is avoided inside the loop. That computation is performed outside the loop as computing the same expression each time was overhead to the system, and this reduces computation overhead and hence optimizes the code.

Example:

Before optimization:

```
for (int i=0; i<10;i++)
t= i+(x/y);
```

```
...
end;
```

After optimization:

```
s = x/y;
for (int i=0; i<10;i++)
t= i+ s;
...
end;
```

6. Loop Unrolling

Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations. Loop unrolling increases the program's speed by eliminating loop control instruction and loop test instructions.

Example:

Before optimization:

```
for (int i=0; i<5; i++)
   printf("Pankaj\n");
```

After optimization:

```
printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
printf("Pankaj\n");
```

7. Loop Jamming

Loop jamming is combining two or more loops in a single loop. It reduces the time taken to compile the many loops.

Example:

Before optimization:

```
for(int i=0; i<5; i++)
   a = i + 5;
for(int i=0; i<5; i++)
   b = i + 10;
```

After optimization:

```
for(int i=0; i<5; i++)
```

```
{
        a = i + 5;
        b = i + 10;
}
```

8. Loop Fission

Loop fission improves the locality of reference, in loop fission a single loop is divided into multiple loops over the same index range, and each divided loop contains a particular part of the original loop.

Example:

Before optimization:

```
for(x=0;x<10;x++)
{
  a[x]=…
  b[x]=…
}
```

After optimization:

```
for(x=0;x<10;x++)
   a[x]=…
for(x=0;x<10;x++)
   b[x]=…
```

9. Loop Interchange

In loop interchange, inner loops are exchanged with outer loops. This optimization technique also improves the locality of reference.

Example:

Before optimization:

```
for(x=0;x<10;x++)
for(y=0;y<10;y++)
a[y][x]=…
```

After optimization:

```
for(y=0;y<10;y++)
for(x=0;x<10;x++)
a[y][x]=…
```

10.Loop Reversal

Loop reversal reverses the order of values that are assigned to index variables. This help in removing dependencies.

Example:

Before optimization:

```
for(x=0;x<10;x++)
a[9-x]=…
```

After optimization:

```
for(x=9;x>=0;x--)
a[x]=…
```

11.Loop Splitting

Loop Splitting simplifies a loop by dividing it into numerous loops, and all the loops have some bodies but they will iterate over different index ranges. Loop splitting helps in reducing dependencies and hence making code more optimized.

Example:

Before optimization:

```
for(x=0;x<10;x++)
if(x<5)
a[x]=…
else
b[x]=…
```

After optimization:

```
for(x=0;x<5;x++)
a[x]=…
for(;x<10;x++)
b[x]=…
```

12.Loop Peeling

Loop peeling is a special case of loop splitting, in which a loop with problematic iteration is resolved separately before entering the loop.

Before optimization:

```
for(x=0;x<10;x++)
if(x==0)
a[x]=…
```

```
else
b[x]=…
```

After optimization:

```
a[0]=…
for(x=1;x<100;x++)
b[x]=…
```

13. Unswitching

Unswitching moves a condition out from inside the loop, this is done by duplicating loop and placing each of its versions inside each conditional clause.

Before optimization:

```
for(x=0;x<10;x++)
if(s>t)
a[x]=…
else
b[x]=…
```

After optimization:

```
if(s>t)
for(x=0;x<10;x++)
a[x]=…
else
for(x=0;x<10;x++)
b[x]=…
```