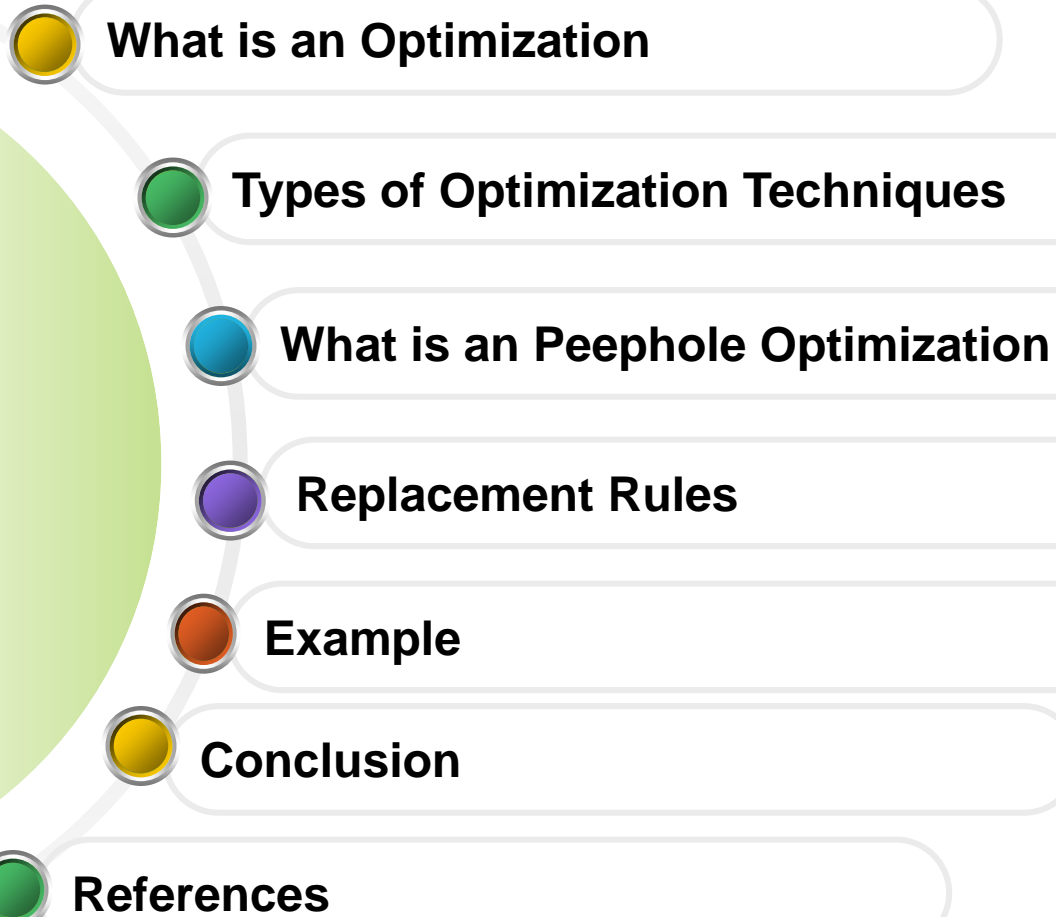


Peephole Optimization Techniques In Compiler Design

Anul Chaudhary

131CC00304

Contents

- 
- What is an Optimization**
 - Types of Optimization Techniques**
 - What is an Peephole Optimization**
 - Replacement Rules**
 - Example**
 - Conclusion**
 - References**

What is Optimization?

Optimization is the process of transforming a piece of code to make more efficient (either in terms of time or space) without changing its output or side-effects. The only difference visible to the code's user should be that it runs faster and/or consumes less memory. It is really a misnomer that the name implies you are finding an “optimal” solution— in truth, optimization aims to improve, not perfect, the result.

Types of Optimization

Optimization can be categorized broadly into two types :

1. Machine Independent
2. Machine dependent

Machine-independent Optimization

The compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations. For example

```
do {  
    item = 10;  
    value = value + item ;  
    } while(value<100);
```

This code involves repeated assignment of the identifier item, which if we put this way:

```
Item = 10;  
do  
{  
    value = value + item ;  
} while(value<100);
```

It should not only save the CPU cycles, but can be used on any processor.

Machine-dependent Optimization

Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references. Machine-dependent optimizers put efforts to take maximum advantage of memory hierarchy.

Peephole Optimization

Peephole Optimization is a kind of optimization performed over a very small set of instructions in a segment of generated code. The set is called a "peephole" or a "window". It works by recognizing sets of instructions that can be replaced by shorter or faster sets of instructions.

Goals:

- improve performance
- reduce memory footprint
- reduce code size

Replacement Rules

Common techniques applied in peephole optimization:-

- ▶ **Constant folding** – Evaluate constant sub-expressions in advance.
- ▶ **Strength reduction**– Replace slow operations with faster equivalents.
- ▶ **Null sequences** – Delete useless operations.
- ▶ **Combine operations** – Replace several operations with one equivalent.
- ▶ **Algebraic laws** – Use algebraic laws to simplify or reorder instructions.
- ▶ **Special case instructions** – Use instructions designed for special operand cases.
- ▶ **Address mode operations** – Use address modes to simplify code.

Examples

Elimination of redundant loads and stores

$r2 := r1 + 5$

$i := r2$

$r3 := i$

$r4 := r3 \times 3$

becomes

$r2 := r1 + 5$

$i := r2$

$r4 := r2 \times 3$

Constant folding

$r2 := 3 \times 2$

becomes

$r2 := 6$

Constant propagation

$r2 := 4$		$r2 := 4$		$r3 := r1 + 4$
$r3 := r1 + r2$	becomes	$r3 := r1 + 4$	and then	$r2 := \dots$
$r2 := \dots$		$r2 := \dots$		

$r2 := 4$		$r3 := r1 + 4$		
$r3 := r1 + r2$	becomes	$r3 := *r3$	and then	$r3 := *(r1+4)$
$r3 := *r3$				

$r1 := 3$		$r1 := 3$		$r1 := 3$
$r2 := r1 \times 2$	becomes	$r2 := 3 \times 2$	and then	$r2 := 6$

Copy propagation

$r2 := r1$		$r2 := r1$			
$r3 := r1 + r2$	becomes	$r3 := r1 + r1$	and then	$r3 := r1 + r1$	
$r2 := 5$		$r2 := 5$		$r2 := 5$	

Strength reduction

$r1 := r2 \times 2$	becomes	$r1 := r2 + r2$	or	$r1 := r2 \ll 1$
$r1 := r2 / 2$	becomes	$r1 := r2 \gg 1$		
$r1 := r2 \times 0$	becomes	$r1 := 0$		

References

- ▶ https://en.wikipedia.org/wiki/Peephole_optimization
- ▶ <http://www.iosrjournals.org/iosr-jce/papers/Vol9-Issue4/N0948086.pdf?id=255>
- ▶ <https://class.coursera.org/compilers/lecture/76>

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look. The shapes are concentrated on the left and right sides of the slide, framing the central text.

Thank You