Design and Implementation of Peephole Optimization Based on Extensible Template

ZHANG Hong-Guang
Department of Computer Science and Technology
NanKai University
Tianjin, China
zhanghg@nankai.edu.cn

LAN Xu-Ze
Department of Computer Science and Technology
NanKai University
Tianjin, China
xuzelan@gmail.com

***Abstract*—Optimization functions are crucial to compiler design. Generally, compiler optimization divides into machine-independent optimization based on the intermediate language and machine-dependent optimization based on the object code. In order to make use of the characteristics of the target machine, this paper presents an optimization scheme about peephole optimization based on extensible template, and introduces the implementation of this scheme for a C compiler of 16-bit embedded CPU system.**

Keywords-peephole optimization; pending optimized codes; peephole scope;extensible template;

I. INTRODUCTION

Because of the sensitivity of embedded system for hardware resources, the design of the embedded software has the strict limit to the time and the space. In order to generate high-quality object codes, the optimization of compiler in the embedded system plays a very important role.

In compiler theory, 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 do not actually do anything, or that can be replaced with a leaner set of instructions [1].

II. DESIGN OF PEEPHOLE OPTIMIZATION BASED ON EXTENSIBLE TEMPLATE

Eliminating the flaws of traditional peephole optimization that the optimized instructions must be continuous, we put forward a scheme about peephole optimization based on extensible template. The main idea of this scheme is: By scanning the target codes, the codes which meet the requirements of the template are replaced with the optimized codes. The scheme has the following main advantages.

- The implementation is relatively simple and the serviceability is very strong.
- The expansion of the template is convenient and a unified optimization program is adopted.
- The scope of optimization is not confined to the continuous instructions, and the "hole" is extended as far as possible.
- In order to generate the optimized codes, the codes in the peephole are looped through until no optimization happens.

A. Storage form of pending optimized codes

Pending optimized codes are the assembly codes generated by C compiler. Each instruction in assembly codes occupies a row. We use a doubly-linked list to store these codes. An assembly instruction is stored in a unit of the doubly-linked list. At the same time multiple instructions that participate in the optimization are linked by special pointers so that they can be seen as continuous code theoretically.

B. Form of the template

By analyzing the form of the template, we put forward a form of the template, which is interpreted by the same program and makes the optimized codes not confined to the continuous instructions. Since the template has the universal property, the template needs to use the wildcard characters to express. The form of the operand in assembly instructions is limited, the definition and resolution of the wildcard characters are easy to implement. For example, #1 signifies that the first operand will not change in the subsequent operands.

The template divides into the source template and the replacement template. The source template is used to match the codes, and the replacement template is used to replace the codes that are matched successfully. In order to facilitate the operation and the denotation, we use the following structure to store the template.

```
#define MAXCODENUM 10
struct templete
{
    int len;
    struct {
            int  iscontinue;
            int  operater;
            char *souretmp;
            char *totmp;
    }u[MAXCODENUM];
};
```

Figure 1. Structure of storing the template

978-0-7695-3557-9/09 $25.00 © 2009 IEEE
DOI 10.1109/ETCS.2009.283

Here we specify that a template contains at most ten assembly instructions. It is convenient to extend the template by macro. *len* indicates that how many assembly instructions are contained, and its value depends on the maximum of the source statements and the replacement statements. Array *u* contains the operations of various statements in the template. *iscontinue* shows that whether the next instruction must be linked together with this one to optimize, value 1 shows that the instructions must be linked together, value 0 shows that the instructions do not need to be linked together, value -1 shows that the template is over, value -2 shows that the template is over and the subsequent instructions do not need to be optimized in terms of this template. *sourcetmp* points to the source statement and *totmp* points to the target statement. *operater* says what operation is performed on the instructions in the template. In order to perform the replacement, we define an array which records the specific operator of the assembly source statements corresponding to each wildcard character.

C. *Determination of peephole scope*

Peephole scope directly determines the effect of the optimized program. Peephole can not be extended to the entire function since the jump instruction and function call would bring the uncertain factors, which affect the outcome of the program. In general, it is safe and reasonable that the peephole scope is based on basic block assembly instructions.

III. IMPLEMENTATION OF THE DESIGN SCHEME

A. *Structure of peephole optimization based on extensible template*

The three key points are described hereinbefore. In the next sections we will implement the design scheme in terms of these key points. The Structure drawing is shown in figure 2.

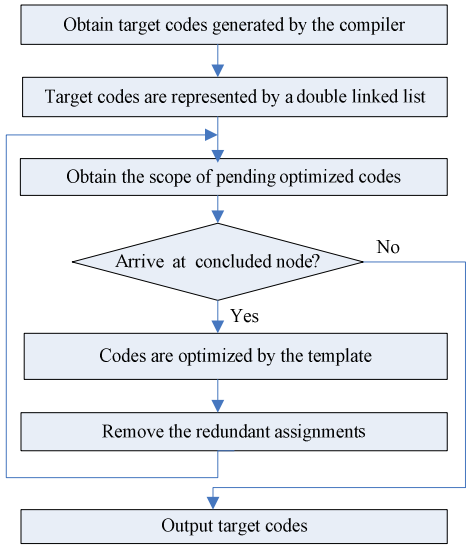

Figure 2. Structure drawing

As shown in Figure 2, the peephole optimization consists of four modules; the following functions are completed by them.

- The pending optimized instructions are represented in the form of doubly linked list in which individual instruction is a storage unit.
- According to the rules of determining peephole scope, the starting node and the ending node are determined for each peephole scope.
- For the statement in the same peephole scope, the proper template array is selected according to the start operator. All possible templates in the array are matched in turn. The codes in the peephole scope are optimized until no optimization occurs.
- Delete the redundant assignments specific to the temporarily registers. In the process of peephole optimization, it is possible to appear immediate operand optimizations, which make the assignment to the temporarily register redundant. Consequently, the redundant assignments are removed after the peephole optimization.

The last two modules are the critical parts, their implementations decide the extent of optimization.

B. *Policies of implementation scheme*

To generate the high-quality target codes, the following policies are adopted.

- All the possible proper templates are matched for each set of assembly instructions.
- When a set of assembly instructions accord with the template and the optimization is performed, the subsequent instructions are traversed in terms of this template until the conditions are not valid.
- Each assembly instruction is traversed all the time until no optimization occurs.
- Remove the unnecessary assignments due to the peephole optimization.

C. *Description of functions*

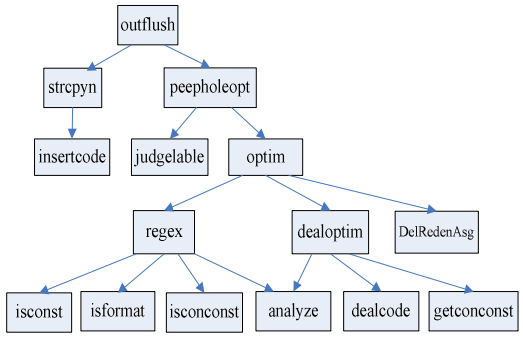

Figure 3. Relation between the functions

Relation between the functions is shown in the figure 3. The target codes generated by the compiler are stored in the form of doubly-linked list by *outflush*, *strcpyn* and *insertcode*. Peephole scope is determined by *peeholetopt* and *judgelabel*. Match and optimization of the codes are

performed by *optim*, *regex*, *dealoptim* and the functions invoked by them. The removal of redundant assignments is completed by *DelRedenAsg*.

Peephole optimization is implemented in the 16-bit embedded CPU system by using the scheme as stated above. This kind of implementation based on the target codes has nothing to do with the structure of the compiler, so it owns universality [2].

IV. TEST AND ANALYSIS

For the moment, we provide C compiler of 16-bit embedded CPU system with 21 templates. To display the optimization effect of the templates, we designedly construct the following function.

```
void max( )
{
  int k,i;
  k = 0;
  i = k++;
}
```

Figure 4. The tested function

The following codes are the target codes generated by C compiler of16-bit embedded CPU system before the peephole optimization is applied. From the comments we can see that the source program is interpreted correctly by the target codes, but the quality of codes is poor.

```
global _max
.text
_max:
mov r5c,sp
sub sp,2
mov r5a,r5c
add r5a,-2+2   // stack position of k
mov r56,0
mov [r5a],r56  // k = 0
mov r5a,r5c
add r5a,-2+2
mov r5a,[r5a]  //fetch the value of k
mov r56,r5c
add r56,-2+2
mov r54,r5a
add r54,1
mov [r56],r54  //do k++ and store it
mov r56,r5c
add r56,-4+2
mov [r56],r5a  /i = k
L1:
mov sp,r5c
ret
```

Figure 5. The target codes

After the peephole optimization is used, the following codes are generated by C compiler.

```
global _max
.text
_max:
mov r5c,sp
sub sp,2
mov r5a,r5c
add r5a,-2+2
mov [r5a],1    // k=1
mov r56,r5c
add r56,-4+2
mov [r56],0    // i=0
L1:
mov sp,r5c
ret
```

Figure 6. The optimized target codes

From the optimized target codes we can see that the optimization effect is obvious. The following templates are used in the example given above.

```
{2, 0,Omit, "mov r%l1, r5c; add r%l1 %n1;#1",0,
-1,Del, "mov r%l1, r5c; add r%l1, %n1",0,0},

{2, 0,Omit, "mov r%l1, r5c; add r%l1 %n1;#1",0,
-1,Rep, "mov r%l2, r5c; add r%l2, %n1; mov %d1, [r%l2]",
"mov %d1, [r%l1]",0},

{2, 0,Omit, "mov r%l1, r5c; add r%l1 %n1;#1",0,
-1,Rep, "mov r%l2, r5c; add r%l2, %n1; mov [r%l2],%*1",
 "mov [r%l1],%*1",0},

{2, 0, Omit,  "mov [%*1], %*2;#1,#2", 0,
-1, Del, "mov %*2, [%*1]", 0,0},
{2, 0, Omit,  "mov %*1, [%*2];#1,#2", 0,
      -1, Del, "mov [%*2], %*1", 0,0},

{2, 0, Omit,  "mov %*1, %*2;#1,#2", 0,
-1, Del, "mov %*2, %*1", 0,0},

{2, 0, Del,  "mov [%*1], %*2;#1", 0,
-2, Omit, "mov [%*1], %*3", 0,0},

{2, 0, Omit,  "mov %*1, 0;#1", 0,
-2, Rep, "add %*1, %*2", "mov %*1, %*2",0}
```

Figure 7. The templates used in the example

Although the forms of the target codes and the optimized target code are different, they all express the meaning of the source program. We use eight templates altogether in the example given above and obtain the efficient target codes.

V. CONCLUSIONS

This paper mainly describes the peephole optimization based on the extensible template in the C compiler of 16-bit embedded CPU system. After a large number of tests, we find that the quality of the target codes is improved greatly by peephole optimization. In the subsequent practice, more templates can be constructed according to the optimized codes and the effect of the peephole optimization is more significant.

REFERENCES

[1] http://en.wikipedia.org/wiki/Peephole_optimization

[2] Jack W.Davidson, Christopher W. fraser. The Design and Application of a Retartgetable Peephole Optimizer. ACM Transaction on Programming Languages and Systems, 1980, 2(2):191~202