第12卷 第4期 华侨大学学报自然科学版 Vol.12 №.4
JOURNAL OF HUAQIAO UNIVERSITY
1991年10月 (NATURAL SCIENCE) Oct.1991

C语言目标代码质量研究*

侯 济 恭
(计算机科学系)

摘要 根据C编译程序的机制,本文以若干翻译实例证明并非所有C语言的目标代都是高质量的,只有那些通晓用抽象和具体(硬件)方式描述问题的程序员才可望获得高质量的目标代码。

关键词 程序设计语言,程序设计,编译程序

0 引言

在各种有关C语言的文章或教科书中,无不推崇其目标码质量[1-4],有些甚至称它可适用一切领域。笔者在剖析*VAX*—11机的C编译程序5.0版(据文献[5、6]和笔者经验知,这是一个较现代的C编译器)发现,上述观点很值得商榷。本文通过分析C编译程序的编译机制,并用若干实例证明:C的目标代码质量高是有条件的,且C并非适用一切领域。

1 C编译程序的编译机制

从宏观上看,C编译程序(下称cc)是一个翻译程序,但从微观上看,它类似于一个解释程序,确切地说,cc是一个类解释型编译程序。略去错误识别,cc的编译过程是:(1)读入一语句行s;(2)若s是符号名说明或定义语句,则填写符号表及分配存贮空间;(3)若s是可执行语句,则:(i)生成关于s的语法t;(ii)对t优化;(iii)遍历优化后之树t',生成目标代码并写入目标文件;(4)重复以上过程直至源程序文件的语句全部处理完毕。

以表达式的识别为例.典型的表达式识别语法分析和语义动作如下

(1)〈语句〉→〈表达式〉*SM*={ecomp($1);}

(2)〈表达式〉→〈表达式$_1$〉〈算符〉〈表达式$_2$〉={$$=buildtree($2,$1,$3);}

* 本文1990—04—10收到

其中 *SM* 定义为分号，"="号后为语义动作。$i 指明产生式右部符 i 个分量，如 $3系指表达式$_2$，$ $要求传递左部的语义值。

函数 buildtree(O,L,r)是建造树，生成二叉树的每一个支，返回结点指针。函数 ecomp(p)编译生成语法树目标代码，其源程序如下

```
ecomp(p) register NODE * p {
/* 入口参数是树结点/根结点指针 */
p=optim(p);/* 对树 p 优化 */
walkf(p,prtdcon);/* 转换树中浮点数 */
locctr( );/* 生成函数头有关代码 */
ecode(p);/* 目标码生成 */
tfree(p);/* 释放树 p */
}
```

例如，对语句

$$y = 5 + x + 12;$$

的翻译过程(图1)如下：(1)生成关于5+x 的子树，设由 *buildtree*，返回的树根指针为 p_1(下同)(2)生成关于 p_1+12的子树，设树根指针为 p_2；(3)生成 y=p_2树，树根指针 p(以上过程，递归调用产生式2)；(4)根据产生式1，执行如下过程：

4.1优化树 p，使成为图2，即合并常数结点5和12，变为17；

4.2生成目标代码：

```
addl3    $17,    -4(fp),    ro ;      ro=x+17
movl3    ro,     -8(fp)        ;      y=ro
```

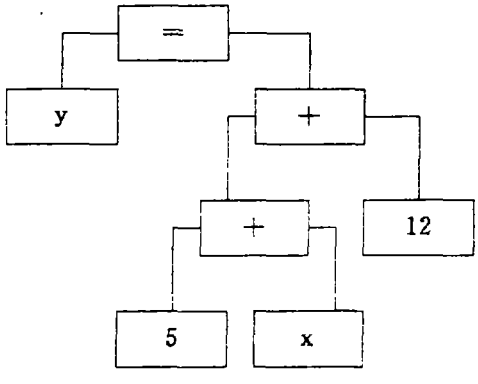

图1 语法树P

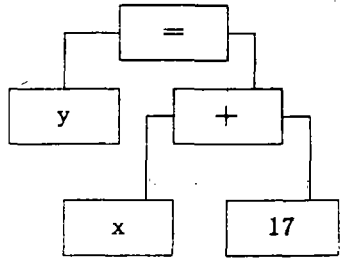

图 2 树P的优化

目标代码质量之高低，关键在于对目标代码或中间代码的优化程度。从上述语法和翻译实例可知，cc 对表达式树(中间代码)的优化仅局限于一棵语法树，即实行的是"窥孔优化"技术。其优化大致有以下5类：(1)合并常数结点：在生成语法树的每一分支时，若左、右子结均为常数

结点便立即计值存入左结，同时释放右结。例如

表达式原形	优化后表达式
x=2*3.14	x=6.28
y=15≪2	y=60
y=7>3	y=1
x=0x57&0xf	y=0x07
a=7>8?b:c	a=c

(2)变换算符：即用运算速度快的算符取代运算慢的算符。如乘2^n运算变成左移运算，除1运算取消，x+(-5)变成x-5等；(3)变换树：以期能合并尽可能多的结点，例如a=2+b*c+8变成a=10+b*c等；(4)关系算符的变换等，如将>改成≤等；(5)进入代码生成阶段，针对具体目标机再调整语法树。显然，以上过程实在称不上"优化"。正如cc的设计者 *S. C. Johnson* 先生所说："Actually the word optimization is something of misnomer, the results are not optimum, only improve。"（事实上称之为优化用词有些不当，其结果并非优化，仅是改良而已。）[7]

影响cc目标码质量的另一个因素是c语言充许递归调用。递归调用意味着每一次调用时都要为局部变量分配存贮地址并在每一次返回后释放该空间[8]。因此，cc编译代码生成策略是尽量避免使用临时寄存器，尽量利用存贮器间的数据直接传递功能，例如x=a=b=c+2的目标代码是

```
addl3   $2,-16(fp),r0      ;r0=c+2
movl    r0,-12(fp)         ;b=r0
movl    -12(fp),-8(fp)     ;a=b
movl    -8(fp),-4(fp)      ;x=a
```

其中fp是栈帧指针，-4，-8，-12，-16分别是x、a、b、c在栈中地址偏移量。显然，该代码若利用寄存器，最后二条指令可变为

```
movl    r0,     -8(fp)
movl    r0,     -4(fp)
```

后者的代码质量显然优于前者。

c的另一个可提高代码效率的功能是寄存器变量。但这也存在下面两个问题：(1)cc事实上是将指令的执行代价分析留给用户，而一般用户，特别是非计算机专业人员是绝无此能力的。(2)cc将若干寄存器划归用户使用(本cc共计5个寄存器归用户使用)，这些寄存器的使用与否，cc一概予以保留。换句话说，cc无判断该资源是否被利用之能力，这实际上是一种资源浪费。

2 实例研究

是否进行循环优化，是衡量目标代码质量的一个重要指标。c具有三种循环类结构语句：

for，while，while—do，使用时一般都是以复合语句的形式出现，即

while(表达式 e){… ;}

do {… ;}while(表达式 e)

for (… ; … ; …){…}

其语法结构是

(3)〈语句〉→〈while 前缀〉〈语句〉|

〈do 前缀〉〈语句〉while(〈表达式〉);|

〈for 前缀〉〈表达式 e'〉)〈语句〉|

〈复合语句〉|〈表达式〉;

其中〈复合语句〉是一分程序体，由一串〈语句〉构成。(、)、;、while 是终结符。

从产生式1、2、3可知，循环类语句的翻译与简单语句的翻译过程相同，只是化整为零进行，不存在对整个循环的任何优化。试以 for 语句翻译为例说明此结论。

```
1:main()
2:{
3:int a,b,i,j,c;
4:int x[20],y[20];
5:a=2;b=3;
6:i=i+b;
7:for(i=1;i<10;i++)
8:      {
9:              j=2;
10:             x[i]=x[i*j]+1;
11:             y[i]=x[i]/j;
12:             a=x[i]+y[i]*b;
13:     }
14: c=a+b;
15: return(c);
16: }
```

相应的目标代码如下列程序所示，各变量地址分配关系如表1所示。

表1 变量名栈内偏移量

变量名	a	b	i	j	c	x	y
栈内偏移量	−4	−8	−12	−16	−20	−100	−180

```
1:          .file   ""                  注释
2:          .data
3:          .text
4:          .align  4
```

```
5:          .globl   main
6:  _main:                                  ;入口
7:          .word    .R1
8:          jbr      .L13
9:  .L14:                                   ;函数体入口。
10:         movl     $2,-4(fp)              ;a=2
11:         movl     $3,-8(fp)              ;b=3
12:         add13    -8(fp),-12(fp),r0      ;r0=i+b   }i=i+b
13:         movl     r0,-12(fp)             ;i=r0     }
14:         movl     $1,-12(fp)             ;i=1
15: .L17:                                   ;循环入口
16:         cmpl     $10,-12(fp)            ;10与i比较
17:         jgeq     .L16                   ;i≥10则 go to L16
18:         movl     $2,-16(fp)             ;j=2
19:         mul13    -16(fp),-12(fp),r0     ;r0=i*j   }
20:         add13    $1,-100(fp)[r0],r0     ;r0=x[r0]+1 }x[i]=x[i]+1
21:         movl     -12(fp),r1             ;r1=i     }
22:         movl     r0,-100(fp)[r1]        ;x[i]=r0  }
23:         movl     -12(fp),r0             ;r0=i     }
24:         div13    -16(fp),-100(fp)[r0],r0 ;r0=x[i]/j }y[i]=x[i]/j
25:         movl     -12(fp),r1             ;r1=i     }
26:         movl     r0,-180(fp)[r1]        ;y[i]=r0  }
27:         movl     -12(fp),r0             ;r0=i     }
28:         movl     -12(fp),r1             ;r1=i     }
29:         mul13    -8(fp),-180(fp)[r1],r1 ;r1=y[i]*b }a=x[i]+y[i]*b
30:         add13    r1,-100(fp)[r0],r0     ;r0=x[i]+r1 }
31:         movl     r0,-4(fp)              ;x[i]=r0  }
32: .L15:
33:         incl     -12(fp)                ;i+1
34:         jbr      .L17                   ;go to L17
35: L16:                                    ;循环终点
36:         add13    -8(fp),-4(fp),r0
37:         movl     r0,-20(fp)
38:         movl     -20(fp),r0
39:         ret#1
40:         ret#1
41:         .set     .R1,0x0
42: .L13:
```

```
43:     movab   -180(sp),sp            ;栈空间保护
44:     jbr     .L14
45:     .data
```

翻译过程大致是:(1)〈for 前缀〉→for(〈表达式 e'〉;〈表达式 e'〉;即识别"for (i<1; i<10;".其主要语义动作是:产生 i=1的目标码(目标代码行14,下同);分配循环终点地址.L15和循环出口地址.L16;产生循环入口地址码.L17(代码行15);产生 i<10的循环出口判断(代码行16、17)。(2)处理产生式3的〈表达式 e'〉,即识别语句"i++;",产生语法树并保存之;再识别右括号,完成对循环说明的处理。(3)〈语句〉→〈复合语句〉:逐句完成循环体内代码翻译,各源语句所对应的代码行是

源语句	目标代码行
j=2	18
x[i]=x[i*j]+1	19~22
y[i]=y[i]/j	23~26
a=x[i]+y[i]*b	27~31

(4)〈语句〉→〈for 前缀〉〈表达式 e'〉〈语句〉:循环终点翻译。产生 .L15(代码行32);将 i++的语法树翻译成目标码(代码行33);产生无条件转至循环入口代码(代码行34)及循环出口地址码.L16(代码行35)。

至此,循环翻译结束。考察目标代码,可以发现六方面缺点。(1)没有删除无用赋值。语句 i=i+b 乃无用赋值,但对应的目标代码行12、13依然保留。(2)没有做代码外提。语句 j=2是循环不变量,但对应的目标码行18仍留在循环体内。(3)没有复写传播。行21,行23,行25,行27,行28均是 r0=i 或 r1=i 即

```
movl -12(fp),r0        或        movl -12(fp),r1
```

完全可用复写传播或保留在一个寄存器内。(4)没有删除多余运算。例如行21至行25,r1值保持不变,重复执行二次 r1=i 即 movl -12(fp),r1指令。同样,行28亦可删除。(5)没有进行运算强度削弱处理,如代码行19

```
mul13 -16(fp), -12(fp),r0
```

即 r0=i*j 可变成

```
add13 -12(fp), -12(fp), r0
```

(6)没有进行数据流分析,否则可以减少许多重复操作。假如再使用多一点寄存器,则代码效率将大大提高。

根据以上分析,可进一步将代码优化(仅对循环代码14行至32行优化)如下:(1)将代码行18:j=2 外提;(2)代码行23:movl 12(fp), r0 改为 movl r1, r0;(3)代码25,27,28行删除;(4)代码行24目标寄存器 r0改为 r2;代码行26源寄存器 r0改为 r2。

经过以上优化,循环体内代码长度由16行减为12行,长度减少25%且强度有所削弱。修改后的目标代码如下

```
1:      .file   ""
2:      .data
```

```
3:              .text
4:              .align  4
5:              .globl  main
6:      _main:
7:              .word   .R1
8:              jbr     .L13
9:      .L14:
10:             movl    $2,-4(fp)
11:             movl    $3,-8(fp)
18:             movl    $2,-16(fp)
14:             movl    $1,-12(fp)
15:     .L17:
16:             cmpl    $10,-12(fp)
17:             jgeq    .L16
19:             mul13   -16(fp),-12(fp),ro
20:             add13   $1,-100(fp)[ro],ro
21:             movl    -12(fp),r1
22:             movl    ro,-100(fp)[r1]
23:             movl    r1,ro
24:             div13   -16(fp),-100(fp)[ro],r2
26:             movl    r2,-180(fp)[r1]
29:             mul13   -8(fp),-180(fp)[r1],r1
30:             add13   r1,-100(fp)[ro],ro
31:             movl    ro,-4(fp)
32:     .L15:
33:             incl    -12(fp)
34:             jbr     .L17
35:     L16:
36:             add13   -8(fp),-4(fp),ro
37:             movl    ro,-20(fp)
38:             movl    -20(fp),ro
39:             ret#1
40:             ret#1
41:             .set    .R1,0x0
42:     .L13:
43:             movab   -180(sp),sp
44:             jbr     .L14
45:             .data
```

3 C的优势

以上分析证明，用非所长，则C语言的代码质量很差。C号称"高级汇编语言"是不无道理的，同时，这一称号也表明了C的优势之所在。概言之，C的优势是：(1)可以干预CPU操作。C拥有强大丰富的位操作功能：&(与)、|(或)、～(非)、^(异或)、≫(右移)、≪(左移)，这些操作与汇编码几乎是一对一关系。

```
int a,b,c,d,e;
a≪b;
        ashl    -8(fp),-4(fp),ro
a≫c;
        mnegl   r0, -12(fp)
        ashl    r0, -4(fp),ro
alb;
        bis13   -8(fp), -4(fp),r0
a&b;
        mcoml   r0, -8(fp)
        bic13   r0, -4(fp),ro
a^ b;
        xor13   -8(fp), -4(fp),ro
～a;
        mcoml   r0, -4(fp)
```

(2)简练的条件表达式，其目标代码质量与手编程序相去无几。如

```
a=b>c?b:c;
            cmpl    -12(fp), -8(fp)
            jleq    .L9999
            movl    -8(fp),r0
            jbr     .L9998
.L9999:
            movl    -12(fp),r0
.L9998:
            movl    r0, -4(fp)
```

(3)赋值型运算(如b+=c)及多重赋值(如a=b=c=1)不仅能降低树的高度，并且能提高代码质量，如

```
  b+=c;
        add12   -8(fp), -12(fp)
a=b= c=1;
        movl    $1, -12(fp)
```

```
movl    -12(fp), -8(fp)
movl    -8(fp), -4(fp)
```

(4)自增与自减运算与汇编指令成一一对应关系，如

```
a++;
    incl    -4(fp)
a--;
    decl    -4(fp)
```

(5)字段操作将机器字结构化，使对字位操作比汇编可简洁，节省存贮空间。(6)巧妙和尽可能充分地利用寄存器，可大大提高代码质量。(7)可用类似于汇编指令中的取地址、间址，间变址等操作来进行地址运算。(8)可直接调用汇编程序段，使汇编与C融成一体。

4 结论

C语言的能力介于高级语言与低级语言之间，它具有高级语言面向抽象的算法描述能力(do、for等)，同时又具有低级语言面向具体的机器硬件描述能力，可谓兼美。C的"窥孔优化"能力有利于系统程序，特别是操作系统型的程序(如定位，调试等)，但不利于数据或数值处理；C的函数参数类型检查太弱有利于系统程序员变换数据类型(如字符与整型运算)，但不利于科学计算(可能产生谬误)；科学型或数据处理注重对问题的抽象描述，对硬件几乎不感兴趣。因此，上文所提到的C的优势对它形同虚设，但这对系统程序至关重要，因为系统程序注重对硬件的描述能力，当然，也注重对问题的抽象描述。若C仅用于抽象描述，C的优势(灵活、高效)无法发挥，若C仅用于硬件描述，抽象描述的优势(节省编程时间，提高程序的可靠性和可读性可维护性等)无法利用。结论是：C语言是系统程序语言，它适用于熟练的系统程序设计人员。

目前，C语言正朝着*PASCAL—C*方向发展，逐步克服其自身的不足。许多新版的C语言增加了"指明优化"能力。从编译理论可知：优化主要是在中间代码级进行，这类优化不依赖具体的目标机，因此具有可移植性。新版CC是老版CC的移植品，故其"指明优化"大都是对目标代码的优化，属于CC的附加品。不过，随着CC与C的发展，C语言将日益完善，最终成为一个真正的通用程序设计语言。

参　考　文　献

〔1〕 顾忠勋译，程序设计语言C，计算机工程与应用，1(1982).

〔2〕 李德华，C语言*BNF*解释及其程序设计，陕西科学技术出版社，(1986).

〔3〕 赖翔飞，CC目标高效的几个关键因素，计算机应用与软件，1(1988).

〔4〕 张乃孝，有关C语言形式化中若干问题，计算机工程与应用，2(1985).

〔5〕 费宗莲，C语言新发展，中国计算机用户，17(1988).

〔6〕 秦学礼译，C语言的美国标准化现状，软件产业，1(1988).

〔7〕 Johnson S. C., *A Tour Throught the Portable C Compiler*, AT&T Bell laborataries, Murray Hill, N.J, (1979).

〔8〕 Writh. N. ,硬件体系结构与程序设计语言的互相支持,计算机科学, 4(1989).
〔9〕 Marejarani, M. ,四种语言 Pascal、C、Lisp、Ada 比较,计算机科学, 5(1989).
〔10〕 王博文、侯济恭,VAX 机 C 编译器分析报告与移植概要,计算机工程与应用, 2(1990).

A Study on the Quality of Object Code Generated by C Compiler

Hou Jigong
(*Department of Computer Science*)

Abstract Based on the mechanism of C compiler, this paper exemplifies that not all the object codes generated by C compiler are of high quality. Only those who are well versed in describing problem with abstract means and concrete means (hardware) can get the object codes of high quality.

Key words programming language, programming, compiler