基于 GCC 的 TMS320C67xx 汇编代码的优化

王 浩[1,2]
(1. 中国科学院长春光学精密机械与物理研究所 长春 130033;
2. 中国科学院航空光学成像与测量重点实验室 长春 130033)

摘 要: 为了使生成的汇编代码具有更高的执行效率,设计并实现了一种基于 GCC 的 TMS320C67xx 汇编指令级的代码优化算法。首先,将汇编指令按照功能划分为不同的指令类型,并将汇编指令链接到链表中。然后,针对每一个寄存器建立对该寄存器的读写操作指令链表。最后,通过对指令类型的判断和对寄存器读写操作指令链表的分析,完成了冗余代码的删除和指令合并。实验结果表明,经过代码优化后,TMS320C67xx 汇编代码的执行效率提高了 20%左右,较中间代码级的优化算法执行效率提高了 15%左右。

关键词: GCC 编译器;DSP;代码优化
中图分类号: TN602 **文献标识码**: A **国家标准学科分类代码**: 510.10

TMS320C67xx assembler code optimization based on GCC

Wang Hao[1,2]
(1. Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun 130033, China;
2. Key Laboratory of Airborne Optical Imaging and Measurement, Chinese Academy of Sciences, Changchun 130033, China)

Abstract: In order to generate higher efficiency assembler code, design and implement a TMS320C67xx assembler instruction level optimization algorithm based on GCC. Firstly, assembler instructions are divided into different types according to the function of instructions. Then, link assembler instructions to the list. Secondly, establish a read and write operation instruction list for each register. Finally, through the judgment of instruction types and the analysis of read and write instructions list for each register, complete the deletion of redundant code and the merging of instructions. The experimental results show that the execution efficiency of TMS320C67xx assembler code increased by about 20% after code optimization. Compared with the intermediate code optimization, execution efficiency increased by about 15%.

Keywords: GCC compiler; DSP; code optimization

1 引 言

新型芯片层出不穷,特别是被用于实现高速运算的数字信号处理器(digital signal processor,DSP)芯片已经在人们的生产、生活的各个领域发挥了重要的作用[1-3]。DSP 芯片的典型代表是 TMS320C67xx 芯片,它具有运算速度快,处理能力强的特点,并且能够处理浮点指令。代码优化是指在不改变程序运行结果的前提下,对代码进行等价的变换。通过变换使得最终生成的目标代码具有更高的执行效率。在经过代码优化后,目标代码的长度缩短了,运行时间缩短了,所占空间减少了。

目前绝大多数的代码优化是在中间代码一级完成的,因为这类优化具有较强的通用性,它不依赖于具体的目标机器。通过对中间代码数据流的分析,引入控制块分解流图可以达到汇编代码优化的目的[4]。在中间代码层设计一个指令调度器,也可以实现汇编代码的优化[5-6]。还可以通过在中间代码层实现条件分支的预测和将循环体展开,完成汇编代码的优化[7-8]。除了中间代码层的优化之外,在汇编代码生成之后,针对汇编指令级的优化也具有其自身的优势。这类优化具有针对性更强,优化更加细致等特点。因此,汇编指令级的优化同样也发挥着重要的作用。

该文在 GCC 编译器生成 TMS320C67xx 汇编代码之后,设计实现了一种汇编指令级的优化算法。算法结合了

收稿日期:2015-01

具体的指令类型，并建立了一个寄存器读写操作指令链表。通过对指令类型的判断和对寄存器读写操作指令链表的分析，完成了冗余代码的删除和指令的合并。在经过代码优化后，汇编代码的执行效率提高了20%左右。

2 代码优化的提出

2.1 GCC的优化策略

GCC在汇编代码生成的过程中做了一些优化，其优化步骤如下：

1)GCC首先对代码进行全局优化。在对源程序进行语义分析和中间代码生成的过程中，会产生一些冗余指令，GCC先删除这部分冗余指令。接下来GCC会做一些公共子表达式的删除，常量/复制传播删除等方面的优化，最大程度地精简程序。

2)全局优化之后，GCC对代码进行的是循环优化。在循环优化的过程中，具体进行了循环不变量外提，强度削弱，基本归纳变量的删除等方面的优化。

3)接下来进行的是基本块优化。基本块优化常常和优化参数联系起来，根据优化参数的不同，执行不同程度的优化。这一部分的优化包括死代码的删除等[9]。

2.2 TMS320C67xx代码优化提出

尽管GCC已经做了很多方面的优化，但针对生成的特定体系结构的汇编代码，仍需要做进一步的代码优化。针对生成的TMS320C67xx汇编代码，有部分代码是冗余的，有部分代码可以用较少的代码来替代。例如，会出现以下情况：

```
1) MVK 9 , A3
   MV A9,B9
   SUB A1,A2,A4
   ADD A1,A2,A3
```

第1条指令MVK 9,A3将立即数9赋值给了寄存器A3,在接下来的两条指令都没有读取A3中的值，第4条指令将A1和A2相加后的结果赋值给了A3,A3中的值被覆盖。A3中原来的值9在A3被第2次赋值之前没有被使用，MVK 9,A3这条指令就是冗余指令。

```
2) B .S2 B3
   NOP 5
   MVK .S1 1, A3
   STW .D1T2 A3, *-A15[1]
```

对于TMS320C67xx的指令集，在无条件跳转指令后面只能在指令的延迟间隙继续执行5个周期，这5个周期全部被NOP 5占用，因此MVK.S1 1,A3和STW.D1T2 A3,*-A15[1]这2条指令将不会被执行，这2条指令是冗余指令。

```
3) MV A0 , B0
   MV B0 , A0
```

A0和B0中的值相等，因此，这2条指令可以合并为MV A0,B0这1条指令。

```
4) MVKL label B3
   MVKH label B3
   B B3
```

通过MVKL和MVKH指令将标号label赋值给B3，再跳转到B3所指向的地址处。TMS320C67xx指令集中的跳转指令支持将一个32位数的标号直接作为目的操作数，所以这3条指令可以合并为B label这1条指令。

出现上述情况，可能是由于GCC在寄存器分配上产生的一些副作用或者GCC之前所做的优化没有结合具体的目标机器指令集。因此，可以设计一个在汇编指令级的代码优化算法，完成冗余代码的删除和指令的合并。

3 TMS320C67xx代码优化

3.1 算法思想

在汇编代码生成之后，算法以一个基本块(一个标号内)内的汇编代码为单位进行处理，将汇编指令划分为不同的指令类型。同时按照寄存器操作数在指令中出现的位置，将汇编指令对寄存器的操作分为读操作和写操作，针对每个寄存器分别建立对该寄存器的读操作指令链表和写操作指令链表。在代码优化算法的执行过程中，通过对每条汇编指令类型的判断，将冗余指令删除，将可以进行合并的指令用较少的指令替代。通过对寄存器读写操作指令链表的分析，完成一些冗余代码的删除。

算法以一个基本块内的汇编代码为单位进行处理，不考虑分支，跳转等复杂情况，充分保证了算法的正确性[10]。同时，一个基本块内的汇编指令数目有限，因此减少了算法在执行过程中在时间和空间上的开销。

3.2 算法执行步骤

1)将TMS320C67xx的汇编指令按照不同的指令功能划分为不同的类型。并将一个标号内的所有汇编指令链接到链表中。例如有如下一段汇编代码：

```
L0:
STW .D2T1  A15, *--B15
MV  .L1X B15, A15
LDW .D2T1  *B15++, A15
ADD .L2  4, B15, B15
```

按照图1形式链接起来。

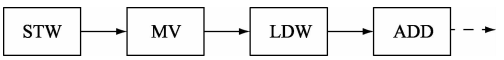

图1 指令链接

每个指令结点结构体包括指令行号、指令的类型、源操作数、目的操作数等成员。指令结点结构体中包含图2所示的指令信息。

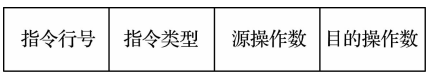

图2 指令信息

2)建立寄存器读写操作指令链表,如图3所示。将TMS320C67xx所有32个寄存器(A_0～A_{15},B_0～B_{15})按顺序编号,每个寄存器分别维护对该寄存器的读操作指令链表和写操作指令链表。链表的结点结构体包含指令所在行号、指令的类型、源操作数、目的操作数等成员,结点按指令所在行号升序链接。

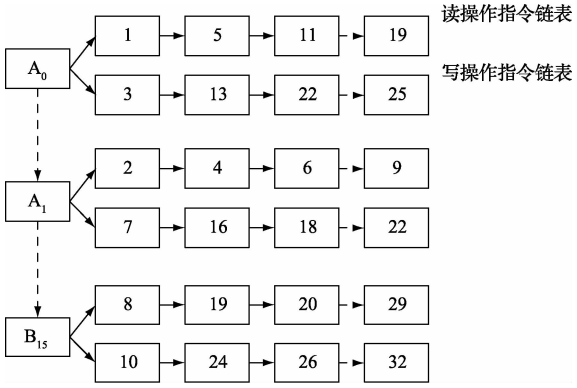

图3 寄存器读写操作指令链表

3)通过判断指令类型实现代码优化。在TMS320C67xx的指令集中,有一类指令将源操作数的值赋值给目的操作数,这一类指令包括MV、MVK、MVKL、MVKH,将它们的指令类型记作mv。mv类指令使用较频繁且具有源操作数值和目的操作数值相等的特点,利用这一特点可以实现指令合并优化。即如果mv类指令的下一条指令将mv类指令的目的操作数作为其源操作数,那么可以将mv类指令的下一条指令的源操作数替换成mv类指令的源操作数,再将mv类指令删除。还可以通过对无条件跳转指令类型的判断,将其5个周期以后的指令删除,同时可以将一个32位的整型标号直接作为跳转类指令的目的操作数,从而达到删除冗余指令的目的。

4)通过对寄存器读写操作指令链表的判断实现代码优化。根据每个寄存器所维护的读写操作指令链表,可以得到指令对该寄存器读写操作的顺序关系,如果在对同一寄存器进行两次写操作之间没有对该寄存器进行读操作,那么对该寄存器的第1次写操作是冗余代码,应该将其删除。

假设在一个标号内,针对寄存器A0,分别有指令所在行号为6,9,18,20的指令对其进行读操作,有指令所在行号为5,11,14的指令对其进行写操作。

read:6 ⟶9 ⟶18 ⟶20

write:5 ⟶11 ⟶14

在对A_0进行写操作的行号为11和行号为14的指令之间,没有对A_0进行读操作的指令,就应该将行号为11的写操作指令删除。算法的具体执行步骤如图4所示。

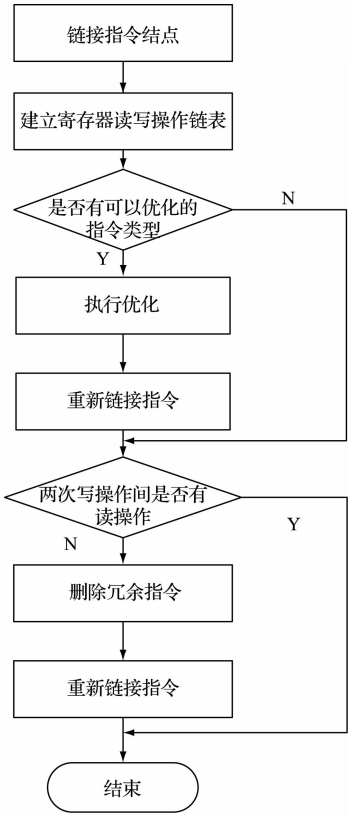

图4 算法执行步骤

首先将TMS320C67xx汇编指令按照不同的指令功能划分为不同的类型,并将一个标号内的指令结点进行链接。然后针对每个寄存器分别建立对该寄存器的读操作指令链表和写操作指令链表。接下来判断是否有可以优化的指令类型,如果有可以优化的指令类型,执行优化操作,再重新链接指令结点。再通过对寄存器读写操作指令链表的判断完成冗余指令的删除,即判断对同一寄存器两次写操作之间是否有读操作,如果没有就删除对该寄存器的第一次写操作指令,再对指令结点进行重新链接。

4 实验测试及分析

实验在Linux操作系统下完成,采用gcc4.0编译器对C语言源程序进行编译,生成TMS320C67xx汇编指令。利用文中所提代码优化算法对生成的TMS320C67xx汇编代码进行优化,并与中间代码级的汇编代码优化算法进行了对比。测试结果如表1所示。

表1 代码优化测试结果

源程序	优化前汇编代码	中间代码优化结果	本文算法优化结果
int main( ) {	main:	main:	main:
int a, b, c,	……	……	……
result, i;	MVK .S2 0, B8	MVK .S2 0, B8	MVK .S2 0, B8
a = 0;	STW .D2T2 B8, *-B14[5]	STW .D2T2 B8, *-B14[5]	STW .D2T2 B8, *-B14[5]
b = 1;	MVK .S2 1, B8	MVK .S2 1, B8	MVK .S2 1, B8
c = 2;	STW .D2T2 B8, *-B14[4]	STW .D2T2 B8, *-B14[4]	STW .D2T2 B8, *-B14[4]
result = add (a, b);	MVK .S2 2, B8	……	……
switch (result) {	STW .D2T2 B8, *-B14[3]	MVKL .S2 add, B8	B .S2 add
case 0:	……	MVKH .S2 add, B8	NOP 5
a += 1;	MVKL .S2 add, B8	B .S2 B8	L4:
break;	MVKH .S2 add, B8	NOP 5	MV .L1 A4, A0
case 1:	B .S2 B8	L4:	……
a += 2;	NOP 5	MV .L1 A4, A0	[B0] B .S2 L6
}	L4:	……	……
for (i = 0; i	MV .L1 A4, A0	[B0] MVKL .S2 L6, B8	L6:
<10; i++)	……	[B0] MVKH .S2 L6, B8	ADD .L1 1, A0, A0
a += 1;	[B0] MVKL .S2 L6, B8	[B0] B .S2 B8	STW .D2T1 A0, *-B14[5]
return 0;	[B0] MVKH .S2 L6, B8	……	B .S2 L5
}	[B0] B .S2 B8	L6:	NOP 5
	……	ADD .L1 1, A0, A0	L5:
	L6:	STW .D2T1 A0, *-B14[5]	MVK .S2 0, B8
	MVK .S1 1, A1	B .S2 L5	STW .D2T2 B8, *-B14[1]
	ADD .L1 A1, A0, A0	NOP 5	L9:
	STW .D2T1 A0, *-B14[5]	L5:	LDW .D2T1 *-B14[5], A0
	B .S2 L5	MVK .S2 0, B8	NOP 4
	NOP 5	STW .D2T2 B8, *-B14[1]	……
	L5:	L9:	L8:
	MVK .S2 32, B8	LDW.D2T1 *-B14[1], A0	LDW .D2T1 *-B14[1], A0
	MVK .S2 0, B8	NOP 4	NOP 4
	STW .D2T2 B8, *-B14[1]	LDW .D2T1 *-B14[5], A0	CMPEQ .L2X 9, A0, B0
	L9:	NOP 4	[! B0] B .S2 L9
	LDW.D2T1 *-B14[1], A0	……	……
	NOP 4	L8:	
	LDW .D2T1 *-B14[5], A0	LDW .D2T1 *-B14[1], A0	
	NOP 4	NOP 4	
	……	CMPEQ .L2X 9, A0, B0	
	L8:	[! B0] MVKL .S2 L9, B8	
	LDW .D2T1 *-B14[1], A0	[! B0] MVKH .S2 L9, B8	
	NOP 4	[! B0] B .S2 B8	
	CMPEQ .L2X 9, A0, B0	……	
	[! B0] MVKL .S2 L9, B8		
	[! B0] MVKH .S2 L9, B8		
	[! B0] B .S2 B8		
	……		

优化前和优化后程序执行周期数对比结果如表2所示。

表2 优化前后执行周期数对比

优化前执行周期数①	214
中间代码优化后执行周期数②	201
本文算法优化后执行周期数③	172
(②－③)/ ②	14.43%
(①－③)/ ①	19.63%

5 结 论

设计并实现了一种通过判断指令类型,分析寄存器读写操作指令链表的汇编代码优化算法。算法完成了冗余代码的删除和指令的合并,经过优化后,代码的执行效率提高了20%左右。GCC大部分的代码优化是在中间代码一级完成的,而本章所提出的汇编指令级的代码优化算法具有更强的针对性,优化更加细致的特点。对其他编译器实现汇编代码优化具有一定的借鉴意义。

参 考 文 献

[1] 刘明亮,朱江森.数字信号处理对电子测量与仪器的影响研究[J].电子测量与仪器学报,2014,28(10):1041-1046.

[2] 韩军,景彩云,吴玲玲,等.基于DSP的图像处理在转角测试中的应用[J].国外电子测量技术,2013,32(1):29-32.

[3] 宋执环,杜往泽,李斌,等.基于图像检测的除尘风机嵌入式控制系统[J].仪器仪表学报,2014,35(5):1192-1200.

[4] 张滇.基于GCC的中间代码优化技术研究[D].哈尔滨:哈尔滨理工大学,2008.

[5] 董锐,王志君,梁利平.基于数据流的指令调度器的设计与实现[J].微电子学与计算机,2011,28(11):148-156.

[6] 王正华.可配置TTA处理器编译器的指令调度技术研究与实现[D].天津:天津大学,2010.

[7] 朱嘉风.面向SIMD的编译指导与条件分支的编译优化技术[D].郑州:解放军信息工程大学,2011.

[8] 余小喜.面向嵌入式系统的迭代式循环展开优化[D].长沙:国防科学技术大学,2011.

[9] BERNHARD S, ZHANG C, CRISTINA C. User-input dependence analysis via graph reachability[C]. 8th IEEE International Working Conference on Source Code Analysis and Manipulation, 2008:25-34.

[10] 胡敏. 基于LLVM编译架构的CSKY后端移植[D].杭州:浙江大学,2014.

作 者 简 介

王浩,1986年出生,硕士研究生,实习研究员。主要研究方向为嵌入式系统开发、图像处理等。
E-mail:wanghao7600@163.com