基于 IMPACT 的嵌入式汇编设计与实现①

冯玉谦, 郑启龙, 卢世贤, 陈思灵, 付和萍

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)
(安徽省高性能计算重点实验室, 合肥 230026)

摘 要: DSP 的底层特性与传统的 C 语言特性差别很大, 有 DSP 领域的特殊指令集, 这些指令很难被编译器生成, 或者根本不能被编译器生成. 程序员也有需求直接访问底层特性. 为了解决这个问题, 通过对 IMPACT 的基础编译器框架进行拓展实现嵌入式汇编功能模块.

关键词: DSP; 编译器; IMPACT

Design and Implementation of Embedded Assembler Based on IMPACT

FENG Yu-Qian, ZHENG Qi-Long, LU Shi-Xian,CHEN Si-Ling, FU He-Ping

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
(National High Performance Computing Center(Hefei), Hefei 230026, China)

Abstract: DSP low-level features is very different from traditional C language features because of specific instruction set in DSP domain which is difficult to be generated by the complier or couldn't be generated by the complier. The basic framework of the IMPACT is expanded to support embedded assemble.

Key words: DSP; complier; IMPACT

1 引言

BWDSP100 是一款国产具有自主产权的高端 DSP 芯片. BWCC 是基于可重定位编译器 IMPACT[1] 研制的 BWDSP100 C 语言编译器. 程序员在进行 C 语言开发时, 往往需要利用嵌入式汇编来访问底层体系结构的一些特性, 用嵌入式汇编实现特定功能的系统函数, 或者利用嵌入式汇编来进行程序调优. 如 ADI 公司的信号处理器 TigerShark 的专用编译器提供了 asm 结构嵌入式汇编. BWCC 通过对 IMPACT 进行拓展支持嵌入式汇编功能.

本文在编译基础框架 IMPACT 下, 设计并实现了针对 BWCC 的嵌入式汇编功能. 其余部分组织如下: 第 2 节阐述了 BWCC 基础框架. 第 3 节阐述了 BWDSP100 体系结构. 第 4 节描述了嵌入式汇编的语法和使用. 第 5 节阐述了嵌入式汇编对编译器前端和后端的影响. 第六部分对全文进行了总结.

2 BWCC基础框架

BWCC 是基于 IMPACT 研制的. IMPACT 是由 UIUC 大学研制的可重定位编译器. BWCC 前端是利用 EDG(Edison Design Group)开发的 C 编译器前端, 先将源程序转换为高层的中间代码 Pcode, 在将 Pcode 转换为底层的中间代码 Lcode. BWCC 后端读入 Lcode,进行相应的优化和代码生成. BWCC 后端分为 6 大模块, 分别为指令注释、指令分簇、软件流水[2]、寄存器分配[3]、指令调度[4]和汇编代码生成, 如图 1 所示, 后端的输入是前端生成的与机器无关的 Lcode; 指令注释把 Lcode 指令注释成与 BWDSPl00 处理器相关的指令; 指令分簇是把指令指定在特定的簇上去执行; 软件流水对循环进行调度, 提高循环部分执行效率; 寄存器分配是把虚拟寄存器转换为物理寄存器; 指令调度对非循环指令进行调度, 提高非循环代码的执行效率; 汇编代码生成是生成 BWDSPl00 的目标汇编代码.

① 基金项目:核高基重大专项(2009ZX01034-001-001-002);安徽省自然科学基金(090412068)
收稿时间:2011-12-25;收到修改稿时间:2012-02-21

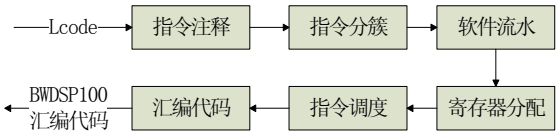

图 1　BWCC 基础框架

3　BWDSP100体系结构

BWDSP100 是一款高性能, 16 发射的 VLIW 结构的浮点运算信号处理器, 有 4 个计算簇, 分别是 X 簇, Y 簇, Z 簇, T 簇, 每个计算簇上有 8 个 ALU, 4 个特殊功能单元, 4 个乘法器. 簇之间通过簇间传输总线通信.

4　BWCC嵌入式汇编语法

通过参考其他编译器(ADI VisualDSP++和 GCC)嵌入式汇编思路, 结合 BWDSP100 的体系结构, 在深入理解 IMPACT 框架的基础上, 制定出 BWCC 嵌入式汇编的接口规范, 如下所示.

嵌入式汇编的语法(接口规范):

asm　[volatile] (

template

[:[constraint(output)[,constraint(output operand)…]]

[:[constraint(input)[,constraint(input operand)…]]

[: clobber]]]

);

嵌入式汇编, 由 4 部分组成, 并使用冒号分隔: 输出操作数列表, 输入操作数列表, 受影响操作数列表(clobber).

具体语法符号定义如下:

Template: 嵌入式汇编的第一部分 template 为一个字符串, 里面描述具体使用的汇编指令.

Constraint: constraint 是一个双引号包含的字符串, 用于指示编译器为输入操作数和输出操作数, 以及分配何种的寄存器.

output operand: output operand 是一些 C 语言的变量名, 下文称之为输出操作数.

input operand: input operand 是一个 C 语言表达式, 下文称之为输入操作数.

Clobber: clobber 表示了可能被汇编指令改写的寄存器.

Constraint 有如下形式:

[use]register class[operation length]

使用选项(use)(详情见表 1)指示编译器是否应该在 template 之前插入指令, 将 C 语言表达式的值拷贝至寄存器中, 以及是否应该在 template 之后插入指令, 将寄存器的值拷贝至 C 语言变量中.

表 1　constraint 中表示使用方法的操作符

Constraint 操作符: 使用方法	
(没有符号说明)	操作数为一个输入操作数
=	操作数为一个输出操作数
+	操作既是一个输入操作数, 也是一个输出操作数
&	如果操作数是一个输入操作数, 那么对应的寄存器不能与任何输出操作数对应的寄存器重合; 如果操作数是一个输出操作数那么对应的寄存器不能与任何输入操作数对应的的寄存器重合

寄存器类型(register class)通过使用表 2 中的符号指明, 编译器根据寄存器类型分配合适的寄存器.

表 2　constraint 中可以表达的寄存器类型

Constraint 寄存器类型	
Constraint	寄存器类型
X	X计算核中的寄存器
Y	Y计算核中的寄存器
Z	Z计算核中的寄存器
T	T计算核中的寄存器
U	U地址生成器中的寄存器
V	V地址生成器中的寄存器
W	W地址生成器中的寄存器

操作长度(operation length)(详情见表 3)告诉编译器对应的寄存器是在普通的定点指令还是在浮点指令中使用.

表 3　constraint 中表示指令操作类型的操作符

Constraint 操作符: 汇编指令操作类型	
(没有符号说明)	用于普通的定点指令
f	用于浮点型的汇编指令中

下图 3 是一个使用嵌入式汇编的 C 语言代码示例, 其中程序定义了 3 个整形变量 a,b,result. 在嵌入式汇编中, %0(result)的 constraint 为"+x", 表示 result 是一个输入和输出操作数, x 表明需要为%0 分配 x 计算核上的一个寄存器. %1(a)的 constraint 为"x"表明要为%1 分配 x 计算核上一个寄存器, a 为一个输入操作数. %2(b)的 constraint 为"x"表明要为%2 为一个 x 计算核上的一个寄存器, b 为一个输入操作数.

```
int result;
int a;
int b;
asm {
“xr19 = %1;”
“xr11= %2;”
“x%0 = %0 + r19;”
“x%0 = %0 + r11;”
: “+x”(result)
: “x”(a), “x”(b)
: “xr19”, “xr11”
}
```

图 3 使用嵌入式汇编的 C 语言代码

5 BWCC嵌入式汇编框架设计

5.1 支持嵌入式汇编的 BWCC 前端设计

支持嵌入式汇编要求编译器前端能正确解析 asm 结构. 前端 EDG 能正确的解析 asm 结构, 并形成相应的抽象语法树结构. 高层的中间代码 Pcode 也支持相应的嵌入式汇编. 上述使用嵌入式汇编的 C 语言对应的 Pcode 文本形式如下图 4 所示.

其中第 1 行表明了嵌入式汇编语句的 template. 第 2 行至第 19 行显示的输入操作数和输出操作数, 图中第 4 行至第 8 行显示变量 result 的约束为”+x”, 第 9 行至第 13 行显示变量 a 的约束为”x”, 第 14 行至第 18 行显示变量 b 的约束为”x”. 第 20 行至第 24 行显示的是嵌入式汇编结构中的 clobber, 图 4 中显示的是寄存器 xr9 和 xr10.

通过对前端的 PToL 阶段进行相应的拓展, 使得将嵌入式汇编对应的 Pcode 形式的高层中间代码转换为相应的底层中间代码 Lcode. 嵌入式汇编从 Pcode 转化为 Lcode 的算法的流程如图 5 所示.

其中, asm 属性的内容为 template 中的字符串, output 的性质描述了编译器为每个输出操作数在 template 中分配的虚拟寄存器号, 以及每个输出操作的约束 constraint; input 的性质描述了编译器为每个输入操作数在 template 中分配的虚拟寄存器号, 以及每个输入操作数的约束 constraint; asm_clobbers 的性质描述了可能被 template 改写的物理寄存器.

```
1. ASM (3 STRING “xr19 = %1; xr11 = %2; x%0
= %0 + r19; x%0 = %0 + r11;” (1 28))
2. OPERANDS
3. (
4. ( 4 ASMOPRD
5. “+”
6. “x”
7. )
8. (5 VAR ID “ result” (1 26))
9. (6 ASMOPRD
10. “ “
11. “x”
12. )
13. (7 VAR ID “a” (1 24))
14. (8 ASMOPRD
15. “ “
16. “x”
17. )
18. (9 VAR ID “b” (1 25))
19. )
20. CLOBBERS
21. (
22. (1 STRING “xr9” (1 22))
23. (2 STRING “xr10” (1 22))
24. )
```

图 4 嵌入式汇编在 Pcode 中的表示

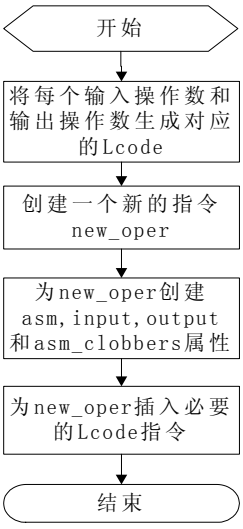

图 5 嵌入式汇编 PToL 算法流程

上述使用嵌入式汇编的 C 语言对应的 Lcode 文本形式如下图 6 所示. 其中第 1 行表示的是嵌入式汇编程序所在的控制块 cb, 第 2 行到第 4 行是对程序中三个变量的处理, 第 5 行表示的是嵌入式汇编部分的信息, 包括具体的代码部分和输出操作数列表, 输入操作数列表, 受影响操作数列表, 第 6 行是保存相应的结果, 第 8 行是决定控制块 cb 的跳转.

```
1 (cb 2 0.000000 [(flow 1 4 0.000000)] )
2 (op 11 mov [( r 4 i )][ ( r 3 i ) ])
3 (op 13 mov [( r 5 i )][ ( r 1 i ) ])
4 (op 14 mov [( r 6 i )][ ( r 2 i ) ])
5 (op 10 instrinsic [] [] <( valotile ( i 1 )
   ( asm ( s_1_abs "xr19 = %1; xr11 = %2;
x%0 = %0 + r19; x%0 = %0 + r11;"))
      ( output ( r 4 i ) ( s_1_abs "x" ) ( s_1_abs "@+" ))
   ( intput ( r 5 i ) ( s_1_abs "x" ) ( s_1_abs "@@" ) ( r 6
i ) ( s_1_abs "x" )
  (   s_1_abs "@@" ))
  (   asm_clobbers   (   s_1_abs   "xr19"   )   (      s_1_abs
   "xr11" ) ) ) > )
6 ( op 12 mov   [ ( r 3 i ) ] [ ( r 4 i ) ] )
7 ( op 15 mov   [ ( mac $P16 i ) ][ ( i 0 ) ] )
8 ( op 16 jump [] [ ( cb 4 ) ] )
```

图 6　嵌入式汇编在 Lcode 中的表示

5.2 支持嵌入式汇编的 BWCC 后端设计

嵌入式汇编对指令分簇, 寄存器分配和指令调度阶段有明显的影响, 需要对这三个模块进行相应的拓展来支持嵌入式汇编功能.

5.2.1 支持嵌入式汇编的分簇模块设计

BWCC 所使用的分簇算法是基于寄存器压力的 VLIW DSP 分簇算法[5]. 算法的流程如图 7 所示. 由于嵌入式汇编语句的 output 和 input 性质所描述的虚拟寄存器已经在程序中被程序员通过操作数的约束指定到指定簇中. 因此在进行虚拟控制块进行分簇时考虑操作数的簇信息是否在汇编语句的 input 性质或者 output 性质中被指定, 若已经指定, 则根据指定的簇信息进行分簇, 若未指定, 则根据基于寄存器压力的分簇算法进行分簇.

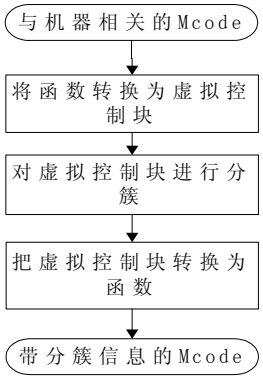

图 7　分簇算法的流程

5.2.2 支持嵌入式汇编的寄存器分配模块设计

BWCC 采用的是 IMPACT 提供的经典的图着色[6] (R.E.Hank. 1993)方法来进行寄存器分配. 该算法主要分为三步, 第一步为构建冲突图, 第二步为图着色, 第三步为插入溢出代码.

①构建冲突图的过程即为对每个宏寄存器进行生成对应的虚拟寄存器, 再对每条指令计算源操作数目和目的操作数目. 加入嵌入式汇编后, clobber 里存放的是真实的物理寄存器, 所以先为这些物理寄存器生成相应的虚拟寄存器, 并为生成的虚拟寄存器完成寄存器分配.

②图着色的过程主要是根据优先级进行迭代直至所有虚拟寄存器都被映射到实际的物理寄存器. 若不能满足这个要求时, 则需要进行虚拟寄存器的溢出, 并修改相应的虚拟寄存器的活跃区间. 加入嵌入式汇编后, 修改虚拟寄存器活跃区间时, 需要考虑到嵌入式汇编语句的特征属性.

③插入溢出代码的过程即在 Lcode 中插入必要的溢出代码. 当加入嵌入式汇编后, 在汇编语句前插入 load 指令, 从内存中读取值到确定溢出的寄存器中, 在汇编语句后插入 store 指令, 将确定溢出的寄存器的值写入到内存中.

5.2.3 支持嵌入式汇编的指令调度模块设计

BWCC 指令调度基于 IMPACT 调度模块, 主要调度过程分为三步, 第一步, 建立指令依赖图, 第二步, 计算指令的优先级, 第三步, 利用表调度算法进行调度. 加入嵌入式汇编后, 约定的调度原则:

①嵌入式汇编之前语句不会调度到之后执行;

②嵌入式汇编之后语句不会调度到之前执行.

遵循这个调度原则, 在建立指令依赖图时, 对相

应的建立过程进行扩展, 如图 8 所示.

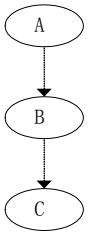

图 8 指令依赖图

其中虚线表示”伪”流依赖, 指令 A, C 对应普通语句, 指令 B 对应嵌入式汇编语句, 则在建立指令依赖图时, A 和 B 建立流依赖关系, B 和 C 建立流依赖关系. 通过建立”伪”流依赖关系来满足嵌入式汇编的指令调度要求.

经过上述拓展后, 上述使用嵌入式汇编的 C 语言对应的汇编形式的目标代码如下图 9 所示.

```
1. xr12= r13
2.|| xr11 = r10
3. xr10 = r14
4. xr19 = r11
5. xr11 = r10
6. xr12=r12 + r19
7. xr12 = r12 + r11
8. xr13= r12
```

图 9 嵌入式汇编被编译后产生的目标代码

编译器为%0 分配 x 计算核上的 12 号寄存器, 为%1 分配 x 计算核上的 11 号寄存器, 为%2 分配 x 计算核上的 x 计算核上的 14 号寄存器,而变量 a 位于 x 计算核上的 10 号寄存器, 变量 b 位于 x 计算核上的 14 号寄存器, 而变量 result 位于 x 计算核上的 13 号寄存器. 若不进行指令的并行调度, 编译器生成的 template 和 template 周围的代码如图 9 所示. 图 9 的 1-3 行将变量 result, a 和 b 的值拷贝至 x 计算核上的 12 号寄存器(%0), x 计算核上的 11 号寄存器(%1)和 x 计算核上的 10 号寄存器(%2); 第 4 行至第 7 行显示寄存器替换后的 template; 第 8 行显示的是将 x 计算核上的 12 号寄存器(%0)的值拷贝至 result 变量.

6 总结

目前主流的 DSP 编译器都提供了嵌入式汇编功能模块, 如 ADI 公司的信号处理器 TigerShark 的专用编译器等, 便于程序中直接调用编译器很难自动生成的 DSP 领域的指令.

本文主要介绍基于 IMPACT 的 BWCC 嵌入式汇编的设计与实现. 通过对 IMPACT 框架的前端阶段和后端, 进行针对嵌入式汇编功能特性的相应拓展来实现嵌入式汇编的功能框架.

参考文献

1 Chang PP, Mahkle SA, Chen WY, Warter NJ, HWU WW. IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors. 18th Annual International Symposium on Computer Architecture. Barcelona: ACM Press, 1998. 408–417.

2 RAUBR. Iterative modulo secheduling: An algorithm for software pipelining loops. Proc. of the 27th International Smposium on Microarchitecture. New York: ACM, 1994. 63–74.

3 Chow F, Hennessy J. Register allocation by priority-based coloring. ACM SIGPLAN Notices, 1984,19(6):222–232.

4 Philip B, Steven P, Gibbons S. Efficient instruction scheduling for a pipelined architecture. ACM SIGPLAN Notices, 1986,21(7):11–16.

5 雷一鸣,洪一,徐云,姜海涛.一种基于寄存器压力的 VLIW DSP 分簇算法.计算机应用,2010,30(1):274–276.

6 Hank RE. Machine Independent Register Allocation for the IMPACT-I C Compiler. Department of Electrical and Computer Engineering, University of Illinois, Urbana IL. 1993.