编译基础设施 Openimpact 调试信息生成的设计①

陈　元[1,2], 郑启龙[1,2], 陈思灵[1,2], 邱鹏飞[1,2]

[1](国家高性能计算重点实验室，合肥 230026)
[2](中国科学技术大学 计算机科学与技术学院，合肥 230026)

摘　要: 基于编译基础设施 Openimpact 开发 DSP 编译器的过程中，调试信息的生成是支持调试功能的必要条件。Openimpact 本身并不支持调试信息的生成，它仅仅将调试信息从源代码携带到前端，以高级中间语言 Pcode 格式存放。为了支持调试信息的生成，我们必须对 Openimpact 进行适当扩展。本文详细讨论了具体扩展方法，包括调试信息从 Pcode 格式到低级中间语言 Lcode 格式的转换以及从 Lcode 格式到汇编格式的生成。

关键词: 调试信息；块作用域；行号；变量

Design of Debugging Information Generation in Compiler Infrastructure Openimpact

CHEN Yuan[1,2], ZHENG Qi-Long[1,2], CHEN Si-Ling[1,2], QIU Peng-Fei[1,2]

[1](Key Laboratory of High Performance Computing, Hefei 230026, China)
[2](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

Abstract: In the process of developping compiler for DSP based on compiler infrastructure Openimpact, the generation of debugging information is necessary for the support of debugging. Openimpact doesn’t support the generation of debugging information which is only carried from source code to front end, stored in the form of advanced intermediate language Pcode. To support the generation of debugging information, it is necessary to make the corresponding expansion on Openimpact. It is discussed in detail that the concrete methods for expansion, containing the conversion of debugging information from Pcode to low-level intermediate language format Lcode and from Lcode format to assembly format.

Key words: debugging information; block scope; line number; variable

1　引言

DSP 编译器的开发一般利用开源的编译器基础设施[1]来支持新的 DSP 体系结构。当前比较常用的开源编译器基础设施有 Openimpact, GCC, LLVM, Open64 等。编译器开发者应根据 DSP 的体系结构特征来选择合适的编译器基础设施，缩短开发时间，并尽可能生成高效的代码。

Openimpact[2]是 EPIC 体系结构的编译器，它的后端有重定向机制，不过后端的大部分工作还是要通过硬编码来实现。GCC 为开发人员做了大量的工作而使得移植过程变得非常简单且高度抽象化，但也正因如此，基于 GCC 的编译器移植缺乏伸缩性和灵活性。

LLVM 在重用 GCC 的前端高级语言处理的同时，采用了自创的代码优化机制，对 GCC 的不足做了大量改进，尤其使得整个程序的全局优化成为可能。Open64 开源编译器的设计结构合理，分析优化全面，是编译器高级研究的理想平台，被许多公司和大学科研项目所采用。

我们要支持的是一款四簇的 VLIW 的 DSP，选择了 Openimpact 开源设施作为基础进行开发，而且要支持 stabs 格式调试信息的生成。

Openimpact 编译架构前端使用的 IR（Intermediate Representation）是抽象语法树形式的 Pcode，后端使用的 IR 是 Lcode [3,4]。Lcode 是一种与机器无关的低级

① 基金项目:核高基项目(2009ZX01034-001-001-002)
收稿时间:2011-09-29;收到修改稿时间:2011-11-12

© 中国科学院软件研究所 http://www.c-s-a.org.cn

中间语言，与大部分 load/store 结构的指令类似，它是一般化的寄存器传递语言。

调试信息的生成是编译器开发过程中的一项重要任务。在嵌入式系统软件开发环境中，调试功能十分重要，对于编译器开发而言，编译器调试信息的生成就成为一个不可或缺的重要功能。

Openimpact 不支持汇编级调试信息的生成，它仅仅是将调试信息从源代码携带到前端，以高级中间语言 Pcode 格式存在。因此，为了支持 stabs 调试信息的生成，必须在 Openimpact 的基础上进行扩展。包括 PtoL 模块（使调试信息从 Pcode 形式转化为 Lcode 形式）、以及调试信息内存管理模块（使 Lcode 形式的调试信息组织成为易管理和易生成 stabs 调试信息的内存格式）、还有对应 Codegen 模块的调试信息打印生成模块。Openimpact 调试信息生成扩展框架图如图 1 所示。

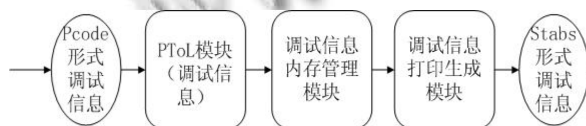

图 1 Openimpact 调试信息生成框架图

本文其余部分组织方式如下。第 2 节介绍调试信息从 Pcode 形式到 Lcode 形式的转换，属于 PtoL 模块。第 3 节介绍从 Lcode 到汇编转换过程，包括调试信息内存管理模块以及打印输出模块。第 4 节是加入调试信息之后的优化效率比较，第 5 节做出结束性总结。

2 调试信息从Pcode到Lcode的转换

2.1 基本流程

程序员在编译时加-g 选项，关于源代码的调试信息，例如行号、局部变量、参数变量、全局变量、块作用域等都应该按照某种格式首先生成到汇编文件中，然后经过汇编、链接，生成含有调试信息段的二进制文件。调试器读取含有调试信息段的二进制文件可以支持单步执行、设置断点、设置观察点等调试功能。

2.2 Stabs 调试信息格式

Stabs[5,6]是一种使用广泛的调试信息格式，在汇编程序中，加入的 stabs 格式调试信息一般是以汇编伪指令的方式出现，它往往与汇编指令语句混杂在一起。比较常用的是两种格式：

.stabs“string”, type, other, dest, value

.stabn type, other, dest, value

通过.stabs 伪指令，可以表示程序的文件名、函数名、变量以及所使用的汇编语言的类型信息等等。而通过.stabn 伪指令则可以表示源程序的行号信息、块结构等。

2.3 行号、变量等调试信息的转化

Lcode 的代码部分自顶向下分别是函数块、控制块、指令、操作数。一个 Lcode 的代码部分由函数块组成。从数据结构上讲，函数块是控制块的双向链表，控制块是指令的双向链表。指令格式按下面给出的格式来组织：

ID, opcode, [dest operands], [src operands]

Lcode 指令中操作数的类型有通用寄存器、立即数、符号标量、字符串和一些特定的寄存器（如 SP，参数传递寄存器等）。

一个函数可以有属性链表，一个指令也可以有属性链表。一个属性可以有一个以上的操作数，方便用于存储扩展信息。在下面的例子中，该属性链表有三个属性组成，分别有一个、两个、一个操作数。我们正是利用 Lcode 的这个特点，从前端到后端传递适当的调试信息。属性链表示例图如图 2 所示。

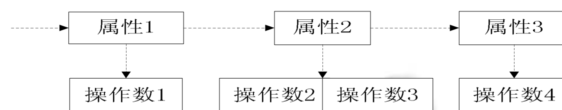

图 2 属性链表示例图

行号信息通过增加指令属性链表的一个属性项来进行传递。增加的这个属性项是行号属性，存储三个操作数：文件名、行号、列号，分别对应这条指令所从属的源文件名、源文件的行、源文件的列。一条源代码一般对应一条或者几条具有相同行号属性项的 Lcode 指令。

局部变量、参数变量等信息以函数属性链表的连续几个属性项保存。一个变量的调试信息一般由四部分组成：变量名、变量类型、存储位置类型（内存变量或是寄存器变量）、具体存储位置（内存偏移量或是寄存器编号）。如果是简单变量（不是数组、指针变量以及函数指针），这四部分就可以用四个属性来表示。对于复杂变量（数组、指针变量或是函数指针），则变量类型部分需要不止一个属性项来表示。故存储一个变量至少需要函数属性链表的四个属性项。

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2.4 块作用域调试信息的传递

2.4.1 Pcode 形式的块作用域

块作用域（block scope）在 Pcode 阶段以抽象语法树复合语句（Compound）的形式存在。整个函数块的块作用域可以用一个 Compound 语句（相当于一棵树的根结点）来表示，一个 Compound 语句有两部分组成：包含在该复合语句内的变量定义和包含在该复合语句内的子语句。其中，子语句既可以是 if、Loop、swith 等类型的语句（这些语句又可以由 Compound 语句组成），也可以是 Compound 语句。Pcode 语句抽象语法树如图 3 所示。

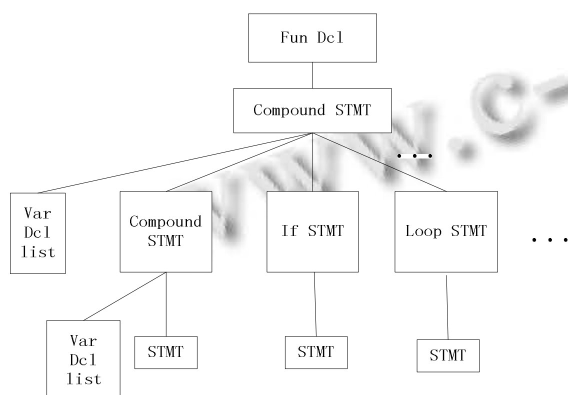

图 3　Pcode 语句抽象语法树

在语义上，一个复合语句就是一个块作用域。块作用域包括变量块作用域和语句块作用域。从汇编调试信息生成的角度讲，变量块作用域研究变量属于哪个块作用域，而语句块作用域研究哪些语句构成一个块作用域。

2.4.2 变量块作用域

由于变量调试信息从前端到后端（PtoL 模块）以函数的属性链表来传递，所有的变量处于一个层次。而在生成变量汇编调试信息时仍然要恢复在 Pcode 阶段时的层次性结构：即变量属于哪一个块作用域，这个块作用域是否属于其它块作用域。

如果仅仅遍历图 3 抽象语法树中的 Compound 语句，将变量属性对应 Compound 语句的属性，则 PtoL 模块以函数属性链表输出的变量次序是先根序遍历此树 Compound 语句结点的变量属性所形成的次序。

可以在遍历此树时把结点对应变量属性的层次信息（块作用域的开始，块作用域的结束，属于哪个块作用域，该块作用域所包含的变量数目等等）输出到一个中间文件中，在调试信息打印输出模块用于指导生成具有层次的变量汇编级调试信息。其中块作用域开始、块作用域标识、变量个数都可以通过先根序遍历此树生成，块作用域结束则可以通过后根序遍历此树生成。

例如：

main:{2;1;{3;1;{4;1;{5;1;}5;}4;}3;{6;1;}6;{7;1;}7;}2;

该层次信息的意思是：main 表示 main 函数的块作用域层次信息，接下去的部分以“;”划分开来，每一个部分为一个基本单元。“{2”表示此处是变量块作用域 2 的开始，“1”表示变量块作用域 2 包含 1 个变量，“}5”表示变量块作用域 5 结束，此时已经按照次序遍历了 4 个变量，以此类推。该层次信息能够保证变量汇编级调试信息层次结构的输出。

2.4.3 语句块作用域

语句块作用域处理的基础也是 Pcode 形式抽象语法树的遍历，核心是找出生成的底层 Lcode 指令与它所属的语句块作用域的关系。先根序遍历图 3 中的抽象语法树，先对此树 Compound 语句结点的作用域进行标识，对于其它树结点，它的作用域标识即为包含它的 Compound 语句的作用域标识。

语句块作用域的传递则类似行号信息的传递。即为每一条指令增加一个块作用域属性，用于保存该指令所从属的块作用域。

2.5 全部变量、自定义数据类型信息的转换

Lcode 的数据部分自顶向下依次是数据链表、数据、表达式。数据由类型信息、地址表达式、值表达式组成。表达式类似于指令的操作数，可以表示一个整数、浮点数、字符串或标签。地址表达式一般为变量的地址标签，值表达式一般表示变量的初始化值。

一个全局变量或静态变量都需要一个数据链表，至少要两个数据项来表示。第一个数据项存储该变量的变量名和变量类型。第二个数据项存储该变量的初始值。

自定义数据类型信息（主要指结构体、共用体）也可以用一个数据链表来表示。例如，一个结构体的定义可以表示为一个数据链表：数据链表的第一个数据项表示结构体的名称，其它数据项表示结构体的成员变量，包括成员变量名、成员变量类型。有几个成员变量，就在该数据链表中依次增添几个数据项。

© 中国科学院软件研究所 http://www.c-s-a.org.cn

3 调试信息从Lcode到汇编的转换

3.1 局部变量、参数变量

变量在函数属性链表中的表示一般需要至少四个属性项，但是通过这种方式表示变量的信息很难使用。可以以变量为基本单元形成新的变量链表，变量的几部分信息是函数属性链表中的连续几个片段，将它们装入变量中组成新的变量链表。这样变量调试信息就从 Lcode 表示形式转换为方便的内存管理形式。变量数据结构变换示意图如图 4 所示。

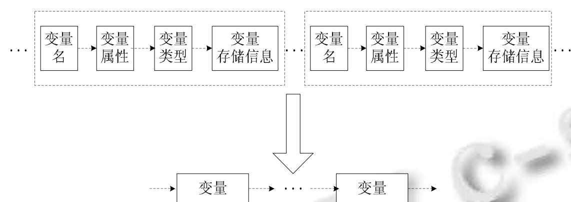

图 4 变量数据结构变换示意图

在寄存器分配阶段，要把寄存器变量从虚拟的寄存器编号修改为实际分配的具体簇的寄存器编号，或者把所有寄存器变量都溢出到内存中。这两种处理方式都需要遍历变量链表去修改对应的变量信息。

在汇编输出阶段从中间文件读取变量块作用域指示信息，依次遍历变量链表，把变量对应的 stabs 格式一一打印输出到汇编文件的对应位置，形成具有层次结构的变量调试信息。

3.2 自定义数据类型

自定义数据类型（结构体、共用体等）也用数据链表来表示，但是这样会使信息分散在几个数据项里，它的 Lcode 表示形式使得随后调试信息的处理非常不方便。故与处理变量调试信息类似，根据 Lcode 形式的自定义类型信息构造一个自定义数据类型链表。

3.3 打印输出主要算法

自定义数据类型调试信息、参变量调试信息、局部变量调试信息和全局变量调试信息在打印输出时，前面定义的变量类型（称为扩展类型，主要包括结构体、共用体、指针、函数指针、一维数组、二维数组、多维数组等）与后面的变量类型引用之间有一个对应关系，这个关系通过二维整数 key=（key1, key2）来标识。一般情况下，key1=0 的 key 标识基本数据类型，key1=1 的 key 标识扩展数据类型。因此，需要实时维护一个 name_key_size 链表来保证这种对应关系。以自定义数据类型调试信息为例说明打印输出算法，参变量调试信息、局部变量调试信息、全局变量调试信息与它类似。

输入：自定义数据类型链表

输出：对应链表中结构体或共用体的类型调试信息和 name_key_size 链表

步骤：

① 依次读取自定义数据类型链表中的元素，如果已到链尾，则退出；

② 先为当前结构体或者共用体生成新的 key 值，再判断该元素是 struct 还是 union，计算其大小，最后把该类型的 name、key 和 size 一并组成元素，插入到 name_key_size 链表中；

③ 依次取当前结构体或共用体成员链表，也就是 field 链表，如果为空则跳转到步骤①；

④ 如果当前成员 field 为简单类型（即不是数组、指针、函数指针等），则可以直接打印输出，跳转到步骤③。如果涉及到多级数组、多级指针、函数指针，则继续；

⑤ 形成所需要的自定义字符数组来抽象表示该数据类型的命名规则。用字符数组记录其依次出现的状态顺序（r, p, f）。r 表示当前维是数组，p 表示当前维是指针，f 表示当前维是函数。根据前面步骤形成的自定义数组名称，查询 name_key_size 链表，如果查询成功则可以打印输出；如果查询失败，则形成 name_key_size 后插入 name_key_size 链表，进行相应打印输出。最后，跳转到步骤①。自定义数据类型调试信息输出算法框架图如图 5 所示。

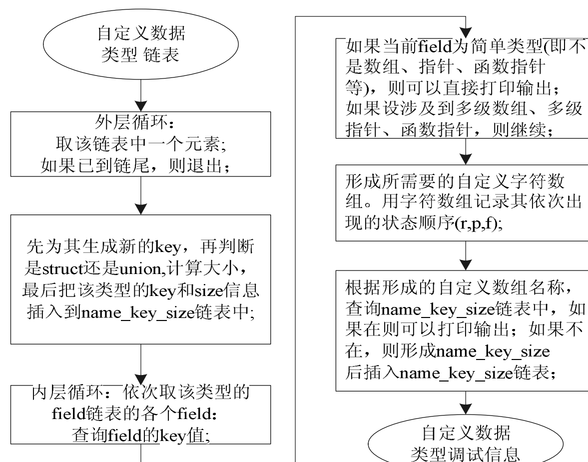

图 5 自定义数据类型调试信息输出算法框架图

© 中国科学院软件研究所 http://www.c-s-a.org.cn

3.4 主要调试信息布局

调试信息布局结构如图 6 所示，依次为源文件路径 stabs、数据类型 stabs、头文件 stabs、函数 stabs、全局变量 stabs。类型信息包括简单数据类型和自定义数据类型。头文件部分依次为表示头文件开始的 stabs、表示函数名的 stabs、表示该函数参量的 stabs、表示头文件名的 stabs、行号 stabs、语句作用域 stabs、层次结构的局部变量 stabs、表示头文件结尾的 stabs。函数 stabs 依次为表示函数名称的 stabs、表示该函数参量的 stabs、表示文件名的 stabs、行号 stabs、语句作用域 stabs、层次结构的局部变量 stabs。

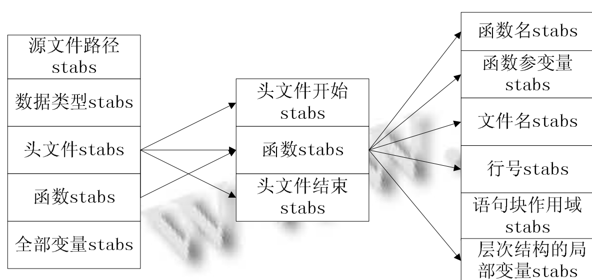

图 6　主要 stabs 调试信息布局图

在打印输出汇编代码时，在调试信息内存管理结构的支持下，依次按照图 6 中得基本布局打印输出汇编代码和调试信息的 stabs 指令。

4　实验

为了验证前述优化方法的性能提升，我们针对 fft 算法在优化前和优化后分别进行了实验，实验结果如图 7 所示。从图中可以看出，相对优化前，时钟周期减少了 40%以上。

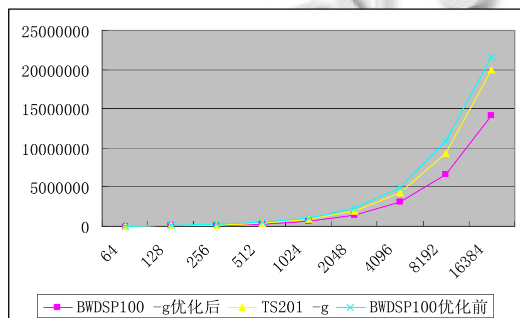

图 7　调试模式优化实验结果

同时我们与 ADSP-TS201 做了性能对比，同样的算法被运行在 TS201 的模拟器上并测量出其执行所用的时钟周期。ADSP-TS201 是美国模拟器件（ADD）公司推出的一款高性能处理器，其处理器架构与我们的 BWDSP100 非常相似。从图 7 中可以看出，在调试模式下BWDSP100编译器相比TS201获得了很大的提升，特别是在 16384 点的时候生成的汇编代码执行时钟周期数比 TS201 减少了近一半。

5　结语

本文在编译基础设施 Openimpact 基础上，设计了支持 stabs 调试信息生成的模块：依次为调试信息 PtoL 模块、调试信息内存管理模块、调试信息打印输出模块。主要介绍了在 openimpact 的基础上进行扩展的具体方法以及主要打印输出算法，并给出了-g 调试模式下与优化前对照比较的实验结果。

参考文献

1 戴桂兰,张素琴,田金兰.编译器基础设施中得多目标编译技术探讨.计算机研究与发展, 2003,40(2):311–317.

2 R A Bringmann. Template for code generation development using the IMPACT-I C compile. University of Illinois, Urbana IL, 1992.

3 B R Rau, V Kathail, S Aditya. Machine-description driven compilers for epic processors. Technical Report. HPL-98-40, Hewlett Packard Laboratories,1998.

4 Julia Menapace, Jim Kingdon, David Mackenzie. The “stabs” debug format.

5 Shannon CJ. The IMPACT SC140 Code Generator. University of Illinois, Urbana IL,2002.

6 胡定磊.VLIW DSP 编译器设计及性能与功耗的优化研究[博士学位论文].国防科学技术大学,2006.

© 中国科学院软件研究所 http://www.c-s-a.org.cn