基于Open64的Fortran90程序源源翻译

高 伟 赵荣彩 姚 远 魏 帅
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 源源翻译是一种很有用的编译基础设施,它将高级语言程序转换为语义等价的可再编译的高级语言程序。目前Open64最新版本5.0中的Fortran90源源翻译还不是很完善,其中有两个突出问题:一是不支持动态数组的翻译;二是含有复杂数据结构的程序激进优化后,中间表示出现伪寄存器,源源翻译出错。在研究Open64的翻译流程和中间表示后,应用信息保存的翻译机制解决了动态数组的源源翻译问题和因为中间表示含有伪寄存器而造成的源源翻译错误的问题。测试结果表明,该方法增强了Open64的源源翻译处理能力。

关键词 Open64,源源翻译,动态数组,伪寄存器,Fortran90
中图法分类号 TP314 **文献标识码** A

Source to Source Translation of Fortran90 Based on Open64

GAO Wei ZHAO Rong-cai YAO Yuan WEI Shuai
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract Source to source translation is a very useful part in modern advanced compiler. It translates one programing language to another, which is equal in semantic and can be compiled again. Currently, source to source translation model of the latest Open64 version 5.0 is not consummate. It has to deal with the following two problems in source to source translation model. One problem is now it can't support dynamic array translation in Fortran90, the other problem is intermediate representation contains pseudo-register after aggressive optimizing. After translation process and intermediate representation were researched, information preservation mechanism was introduced to solve the translation problem of dynamic array and pseudo-register. Test results prove that the method can greatly improve the robustness of the source to source translation in Open64.

Keywords Open64, Source to source translation, Dynamic array, Pseudo-register, Fortran90

1 引言

现代编译器大都分为前端和后端。前端把高级程序转换为中间语言,后端的优化和代码生成都是基于中间语言。编译器基于中间语言做大量的分析和与目标平台无关的优化,如过程间分析、循环嵌套优化等,然后生成目标代码。源源翻译就是编译器的输入是高级语言程序,输出是语义等价的高级语言程序。源源翻译的应用广泛,首先它可以将一种高级语言翻译成另外一种高级语言,如Fortran语言翻译为C语言[2]。其次,它可以实现与平台无关的程序分析和优化,如自动并行化。Rose是一款开源的用于程序自动并行化的源源编译器。Rose在中间表示中应用信息保存的方法,并且放弃了部分优化来保证源源翻译的正确性,它能够正确地翻译Fortran90的动态数组,中间表示不会出现伪寄存器。Suif是另一个支持源源翻译的编译器基础研究平台[1],它由Stanford大学开发。与Open64相比,Suif虽然提供的分析和优化有限,但也不能够正确地源源翻译Fortran90程序。开源编译器Gcc一直没有源源翻译的支持。Barbara Chapman等人基于Open64开发了一个自动生成OpenMP程序的源源编译器OpenUH[6],UPC编译器是另一个基于Open64开发的并行C到C的源源编译器[7]。米伟等人应用类型恢复的方法解决了源源翻译类型不一致的问题[1]。目前Open64的Fortran语言源源翻译whirl2f模块是从Rice大学开发的Co-Array Fortran编译器移植过来的[8],虽有些改动,但框架并没有改变,整个whirl2f模块还不够完善。本文的信息保存翻译方法不仅适合Open64,也可用在Suif等其它不支持Fortran90源源翻译的编译器中。

Fortran90除继承Fortran77的语法特性外,还添加了结构体、指针、动态数组等,同时引入了模块和接口。这些面向对象的程序特征增加了程序的可读性和可维护性。越来越多的科学计算程序都采用Fortran90,如在最新发布的SPECMPI2007测试集18个程序中,有13个采用Fortran90[5]。Fortran90程序带来方便的同时也给Open64编译器的源源翻译带来了新问题,主要存在以下两大困难:一是Open64不支持动态数组的翻译;二是在对结构体数组等复杂数据结构进行激进的优化后,中间表示出现伪寄存器,导致翻译结果不正确。

到稿日期:2012-03-26 返修日期:2012-07-16 本文受"核高基"国家科技重大专项(2009ZX01036)资助。
高 伟(1988-),男,硕士生,主要研究方向为先进编译技术,E-mail:yongwu22@126.com;赵荣彩(1957-),男,博士,教授,博士生导师,CCF会员,主要研究方向为先进编译技术、软件逆向工程;姚 远(1972-),男,硕士,副教授,主要研究方向为先进编译技术;魏 帅(1984-),男,博士生,主要研究方向为先进编译技术。

本文首先给出了动态数组的翻译算法；然后应用信息保存翻译算法解决了激进优化后中间表示含有伪寄存器的翻译问题，并在 Open64 的 whirl2f 模块中实现；最后经过大量测试说明，采用上述方法后 Open64 的 Fortran 语言源源翻译处理能力大大增强。

2 动态数组的翻译算法

动态数组是在程序运行过程中动态地分配存储空间，避免了静态数组浪费存储空间的现象。Fortran90 的语法中添加了动态数组[3]，关键字 ALLOCATABLE 用于说明声明的数组是动态数组，声明时只需指定动态数组的维数，而在使用动态数组前需要通过 ALLOCATE 语句为动态数组分配存储空间并指定数组各维的上下界或维展。

```
integer,dimension(:,:),allocatable,::j_global
allocate(j_global(ny_block,nblocks_tot))
```

图 1 声明动态数组 j_global

图 1 中声明了一个动态数组 j_global，关键字 dimension(:,:)表示 j_global 为二维动态数组，使用动态数组 j_gloab 前，通过 ALLOCATE 语句为其申请存储空间并指定数组的维展分别是 ny_block 和 nblock_tot。目前 Open64 还不能正确地翻译 ALLOCATE 语句。图 1 中 allocate 语句的中间表示如图 2 所示，可以看出，此 whirl node(wn)的操作类型是 OPR_CALL[4]，是一个函数调用节点，应该按照函数调用语句进行翻译。但其实它不是一个真正的函数调用语句，因为真正的函数调用语句其中间表示每个孩子节点代表一个参数，如函数调用语句 add(b,c)，其中间表示如图 3 所示，它的两个孩子节点分别代表参数 b 和参数 c。Open64 按照函数调用语句翻译 allocate 语句的结果是完全错误的，因为翻译结果完全不符合 allocate 语句的语法规则。图 1 中动态数组 j_global 的翻译结果如图 4 所示，其中 & 是 Fortran 中的续行标志符。

```
I8I4LDID 36 <1,48,J_GLOBAL> T<4,.predef_I4,4> map_id(964)
I8I4LDID 48 <1,48,J_GLOBAL> T<4,.predef_I4,4> map_id(965)
I8MPY map_id(627)
U4INTCONST 4 (0x4) map_id(628)
I8MPY map_id(629)
I4I8CVT map_id(630)
I4PARM 2 T<4,.predef_I4,4> # by_value map_id(631)
U4U4LDID 8 <1,48,J_GLOBAL> T<8,.predef_U4,4> map_id(966)
U4INTCONST 1 (0x1) map_id(632)
U4BAND map_id(633)
U4PARM 2 T<8,.predef_U4,4> # by_value map_id(634)
U4INTCONST 0 (0x0) map_id(635)
I4PARM 2 T<4,.predef_I4,4> # by_value map_id(636)
U4INTCONST 0 (0x0) map_id(637)
I4PARM 2 T<4,.predef_I4,4> # by_value map_id(638)
U4U4LDID 0 <1,48,J_GLOBAL> T<8,.predef_U4,4> map_id(967)
U4PARM 2 T<8,.predef_U4,4> # by_value map_id(639)
U4CALL 586 <1,54,_F90_ALLOCATE_B> # flags 0x24a map_id(10)
```

图 2 图 1 中 allocate 语句的中间表示

```
U4LDA 0 <2,2,B> T<49,anon_ptr.,4> map_id(4)
U4PARM 1 T<49,anon_ptr.,4> # by_reference map_id(2)
U4LDA 0 <2,3,C> T<49,anon_ptr.,4> map_id(5)
U4PARM 1 T<49,anon_ptr.,4> # by_reference map_id(3)
F4CALL 2174 <1,45,add_> # flags 0x87e map_id(0)
```

图 3 函数调用语句 add (b,c)的中间表示

```
_F90_ALLOCATE_B(%val(((J_GLOBAL%dims(1)%&ext *
J_GLOBAL%dims(2)%ext) * 4_8),&%val(IAND(J_GLOBAL%
fl-ds%assoc,1)),&%val(0),val(0), val(J_GLOBAL%base))
```

图 4 Open64 翻译动态数组 j_global 的结果

Open64 中所有的动态数组翻译后都变成一个结构体，动态数组 j_global 翻译后变成结构体 J_GLOBAL，如图 5 所示。其中成员 BASE 是原来的数组，成员 DIMS 是它的维数。而 DIMS 又是一个结构体，成员 LB 表示动态数组的下限，EXT 表示动态数组的维展。从中间表示和错误翻译结果可以看出，wn 的第 4 个孩子节点是数组名，但 wn 中维数信息并不全，因此不能按照中间表示直接翻译。

```
TYPE J_GLOBAL
    SEQUENCE
    INTEGER*4 :: BASE(:,:)
    TYPE (FLDS) FLDS
    TYPE (DOPE_BND) DIMS(2)
  END TYPE
TYPE DOPE_BND
    SEQUENCE
    INTEGER*4 LB
    INTEGER*4 EXT
    INTEGER*4 STR_M
END TYPE
```

图 5 动态数组 j_global 翻译后的部分声明

Open64 源源翻译的单位是 Program Unit(PU)，从整个 PU 中能够找到动态数组维数的信息。因此采用信息保存的方法，在源源翻译初始化阶段遍历整个 PU，借助 Open64 中的现有数据结构哈希表 HASH_TABLE 和动态数组 DYN_ARRAY 保存动态数组信息，翻译时通过查找保存信息的数据结构解决维数信息缺失的问题。

定义的数据结构如图 6 所示。图 7 给出了保存动态数组信息的算法，它对当前翻译的 PU 进行遍历，保存每个动态数组的数组名和维数信息到 abd 中，保存 PU 中所有的动态数组到 abd_ay 中，利用哈希表 abd_tab 将动态数组和 abd_ay 对应起来。图 8 给出了 allocate 语句的翻译算法，在翻译 allocate 语句时，首先从当前 wn 中取数组名并翻译，然后在哈希表 abd_tab 中取 wn 的维数信息，其中动态数组的维数 num_dim=(数组 bda 元素个数－1)/2，因为 bda 中保存了数组名。Fortran90 的语法规定动态数组某一维缺省数组的下界为 1。从 bda 中取到的是下界和维展，下界可以直接翻译，上界UB=(LB+EXT－1)，这样无论源程序如何声明都可以正确翻译。应用上面方法翻译图 1 中动态数组 j_global 的正确翻译结果如图 9 所示。

```
typedef DYN_ARRAY<WN*> ARRAY_BOUND;
typedef DYN_ARRAY<ARRAY_BOUND*> ABD_ARRAY;
typedef HASH_TABLE<ST*, ABD_ARRAY*> ABD_TAB;
ARRAY_BOUND* abd /*存放每一个动态数组的维数信息*/
ABD_ARRAY* abd_ay /*存放整个 PU 中所有的动态数组*/
ABD_TAB* abd_tab;/*通过动态数组的符号表 ID 查找动态数组*/
```

图 6 用于保存动态数组信息的数据结构

```
Input: A PU which to be translated
Output: abd ,abd_tab, abd_ay
Function: traverse the whole PU,preserve dynamic array's information
```

```
W2F_Allocate_Walk(WN * wn, MEM_POOL * pool){
  switch{
  case : wn is a func_entry operation node
      foreach kid in func_entry do
      W2F_Allocate_Walk(kid, pool);
      endfor
  case : wn is a block operation node
      foreach wn in block do
      W2F_Allocate_Walk(wn, pool);
      endfor
  case : wn is a allocate node
      / * preserve dynamic array's name * /
      ( * abd) = the fourth kid of wn;
  case: wn is a stid operation node
      if (wn is a dynamic array's LB or EXT)
          / * find wn in abd_tab * /
          ABD_ARRAY * abd_ay=abd_tab->Find(st);
          / * abd_tab don't have current dynamic array * /
          if(abd_ay==NULL)
            / * put current dynamic array to abd_tab * /
            abd_tab->Enter(st, abd_ay) ;
          endif
          / * preserve dynamic array's LB or EXT * /
          ( * abd)=the kid of wn;
    endif
  case: wn is other operation node
      break;
  }
}
```

图 7　保存动态数组信息算法

```
Input: allocate stmtment s'wn
Ouput: translation result call_tokens
Function: translate allocate stmtment
WN2F_allocate(WN * wn){
  output(call_tokens, "ALLOCATE(");
  wn * array_name=get the fourth kid of wn
  Translate(call_tokens, array_name)
  output (call_tokens, '(');
  / * find wn in abd_tab * /
  result=abd_tab->Find(WN_st(wn))
  If(reslut==TRUE)
    abd= get wn's dimension information from abd_ay
  endif
  num_dim=(bda->Elements()-1)/2
  foreach 1≤ i ≤ num_dim do
    / * translate the ith dimension'low bound * /
    Translate(call_tokens, ( * bda)[2 * i-1]);
    Output (call_tokens, ':');
    / * caculate the ith dimension'up bound * /
    WN * ub=SUB(ADD(( * bda)[2 * i], ( * bda)[2 * i-1]),
CONST1)
    / * translate the ith dimension'up bound * /
    Translate(call_tokens, ub);
    output (call_tokens, ',');
  endfor
  output (call_tokens, '))');
}
```

图 8　allocate 语句翻译算

```
ALLOCATE(J_GLOBAL% BASE(&1: MAX(J_GLOBAL%
DIMS(1)%EXT),&1:MAX(J_GLOBAL%DIMS(2)%EXT)))
```

图 9　动态数组 j_global 的正确翻译结果

3　中间表示含有伪寄存器的翻译算法

源源翻译通常都是按照中间表示直接翻译，但是对存在大量优化的编译器 Open64 来说，有时直接翻译的结果不正确。如对含有结构体数组等复杂数据结构程序进行激进优化后，中间表示出现伪寄存器 preg，直接翻译会导致翻译结果含有 preg。图 10 是 SPECMPI2007 测试集中程序 pop2 的一个语句，直接翻译结果含有伪寄存器 MISYM_TEMP_3，如图 11 所示，翻译错误。

```
if(present(j_glob))  j_glob=all_block(block_id)%j_glob
```

图 10　pop2 中一个语句

```
IF(loc(J_GLOB) . NE. (0)) THEN
  MISYM_TEMP_3 = loc(ALL_BLOCKS%BASE((BLOCK_ID-
  &(ALL_BLOCKS% DIMS(1)%LB)) +1)%J_GLOB)
  DO F90LI_0_2_PREG=0,J_GLOB%DIMS(1)%EXT +(-1), 1
    J_GLOB%BASE(F90LI_0_2_PREG +1) =&MISYM_TEMP_
    3( F90LI_0_2_PREG +1)
  END DO
END IF
```

图 11　图 10 语句的错误翻译结果

程序的语义是将结构体数组元素的值赋给另一个数组，但翻译结果中出现了伪寄存器变量 MISYM_TEMP_3，MISYM_TEMP_3 表示结构体数组 all_blocks(block_id)%j_glob 的首地址，其中函数 loc 的功能是求目标对象的内存地址并返回。内存地址值是一个 4 字节整数，可按整数对待，并参与整数运算，也可赋予整形指针变量。翻译结果中寄存器变量 MISYM_TEMP_3 是错误的。这里有两种修补方式，方式一是将变量 MISYM_TEMP_3 声明为整形指针，用指针来翻译，如图 12 所示，但这需要更改变量的声明和引用两处，翻译比较复杂，因此采用方式二即信息保存机制的翻译方法。

```
POINTER(MISYM_TEMP_3,j_glob)
IF(loc(J_GLOB) . NE. (0)) THEN
  MISYM_TEMP_3 = loc(ALL_BLOCKS%BASE((&BLOCK_ID
  -(ALL_BLOCKS% DIMS(1)%LB)) +1)%J_GLOB)
  DO F90LI_0_2_PREG=0,J_GLOB%DIMS(1)%EXT +(-1),1
    MISYM_TEMP_3 +=4;
  END DO
ENDIF
```

图 12　声明 MISYM_TEMP_3 为整形指针的翻译方法

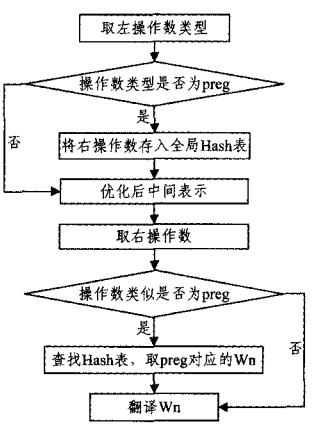

图 13　中间表示含有 preg 的翻译算法框架

信息保存机制的基本思想是利用 Open64 现有数据结构哈希表将 preg 和 wn 对应起来，编译器在激进优化时，如果生成 preg，则将其对应的 wn 保存在哈希表中。在源源翻译时，遇到 preg 时就去哈希表中查找对应的 wn 信息。取出 preg 对应的 wn 进行源源翻译，这样在代码中就没有伪寄存器。算法框架如图 13 所示，利用此算法翻译图 10 所示的语句得到的正确结果如图 14 所示。

```
IF(loc(J_GLOB) .NE. (0)) THEN
  DO F90LI_0_2_PREG=0,J_GLOB%DIMS(1)%EXT+(-1),1
    J_GLOB%BASE(F90LI_0_2_PREG +1) = ALL_BLOCKS%
    BASE ((BLOCK_ID -(ALL_BLOCKS% DIMS(1)%LB)) +
1)%&J_GLOB) (F90LI_0_2_PREG +1))&.
  END DO
END IF
```

图 14　信息保存算法翻译图 10 语句的结果

4　实验评价

实验分为两部分，首先测试动态数组源源翻译的正确性。采用两遍编译方式，即第一遍将源程序进行源源翻译，生成高级语言程序，第二遍再将翻译结果提交基础编译器，然后在测试平台运行，验证其运行结果是否与直接编译源程序的结果一致(认为基础编译器是可靠的)。测试集包括 SPEC CPU 2000、SPEC CPU 2006 和 SPECMPI2007 中所有含有动态数组的 Fortran90 程序，测试用例详见表 1，编译环境为 Linux 操作系统，版本为 ReadhatEnterprise 5，测试环境试验平台的 CPU 为 Intel(R) core(TM)2 T5500，Open64 5.0 作为源源翻译工具，gcc4.1.2 作为基础编译器。分别用-O0，-O2和-O3 作为源源翻译的选项，动态数组源源翻译的正确测试结果详见表 2，测试结果表明，动态数组翻译算法能够正确地翻译测试用例中所有的动态数组。

表 1　动态数组测试用例

测试集	测试用例名称
SPECCPU2000	fma3d，galgel，facerec，lucas
SPECCPU2006	leslie3d，GemsFDTD，tonto，wrf
SPECMPI 2007	Socorro，zeusmp2，l2wrf2，pop2，lu，wrf2，GAPgeofem，lGemsFDTD

实验的第二部分是测试中间表示含有伪寄存器翻译算法的正确性并示源源翻译的作用。我们基于开源编译器 Open64 框架实现自动向量化编译系统 SW-VEC。应用超字并行方法[9]在 Open64 的循环嵌套优化(LNO)中增加自动向量化功能，同时增加数组重组和对齐优[10]。SW-VEC 将源程序翻译成能在国产 CPU SW1600 上运行的带有短向量指令的高级语言程序。实验平台的向量化部件可以同时处理 4 个浮点型数据或者 8 个整形数据。测试用例选用向量化后中间表示含有伪寄存器的程序。实验时首先选用-O3 和-LNO:slp=1 为源源翻译选项，用 SW-VEC 将源程序转化为向量化程序，然后用基础编译器编译成二进制代码并在国产 CPU SW1600 上运行，记录运行结果并将运行时间记为向量化时间；将源程序直接提交给基础编译器，生成二进制代码并在国产 CPU SW1600 上运行，记录运行结果并将运行时间记为串行时间。通过比较两次运行结果验证中间表示含有伪寄存器翻译算法的正确性，用串行时间除以向量化时间得到程序向量化的加速比，图 15 给出了自动向量化的测试结果。测试结果表明，信息保存方法能够正确翻译激进优化后中间表示含有伪寄存器的程序，其中部分程序经过自动向量化后能够取得一定的加速效果。

表 2　动态数组源源翻译正确性测试的结果

源源翻译编译选项	O0+src2src	O2+src2src	O3+src2src
SPECCPU 2000	√	√	√
SPECCPU 2006	√	√	√
SPEC MPI 2007	√	√	√

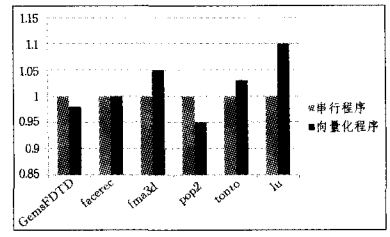

图 15　源源翻译自动向量化的结果

结束语　本文基于 Open64 中间表示 whirl 和翻译流程的研究，给出了 Fortran90 程序动态数组的翻译方法，应用信息保存翻译算法解决了激进优化后中间表示含有伪寄存器的翻译问题，并通过大量的测试验证方法的有效性和健壮性。本文的工作大大增强了编译器 Open64 的 Fortran 语言源源翻译能力。

如何让 Open64 源源翻译出 Fortran90 等面向对象程序是下一步工作。此外，whirl2f 模块还有很多其他的漏洞，因此用形式化方法验证 whirl2f 模块的完备性也是待解决的问题。

参考文献

[1]　米伟，李玉祥，陈莉，等. 带类型恢复的编译器源源翻译技术[J]. 计算机研究与发展，2010，47(7)：1145-1155

[2]　FeldmanS I. A Fortran to C converter[J]. ACM SIGPLAN Fortran Forum，1990，9(2)：21-22

[3]　白云. FORTRAN90 程序设计[M]. 上海：华东理工大学出版社，2003：217-234

[4]　Overview of the open64 Compiler Infrastructure[OL]. http://open64. sourceforge. net

[5]　SPECMPI2007[OL]. http://www. spec. org

[6]　Liao Chun-hua，Hernandez O，Chapman B. OpenUH：An optimizing portable OpenMP compiler[J]. Concurrencyand Computation：Practice and Experience，2007，19(18)：2317-2332

[7]　Chen W. Building a source-to-source UPC-to-C translator [D]. Berkeley：University of California at Berkley，2005

[8]　Rice University[OL]. http://www. rice. edu

[9]　Larsen S，Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets[C]//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. June 2000：145-156

[10]　魏帅，赵荣彩，姚远. 面向 SIMD 的数组重组和对齐优化[J]. 计算机科学，2012(2)