国防科技大学学报
JOURNAL OF NATIONAL UNIFERSITY OF DEFENSE TECHNOLOGY
第12卷第4期 1990年12月 Vol.12 No.4

YFT77向量语句的实现

张可军 郭克榕

(计算机系)

摘 要 本文提出了通过源语言到中间语言的变换进行语言扩充的方法。这种方法已用来在YFT77中扩充向量语句，实践证明，这是一种有效、可行的途径。

关键词 程序语言，源程序，向量语句/中间语言，向量化，语言扩充，程序变换

分类号 TP31

向量语句是YH—1 Fortran编译器(YHFT)的一大特色，它充分利用YH—1多功能流水部件的特点使得YH机的向量速度得以充分发挥，因此很受用户欢迎，现有的YH—1用户程序以及应用库程序中有相当多的向量语句。

与YHFT相比，YH—1的另一个Fortran编译器YFT77具有结构紧凑、编译速度快、标量处理速度快的特点，但其向量处理能力，向量化功能却略弱于YHFT，因此，在YFT77中加入向量语句既是在YFT77中吸收YHFT优点的方法之一，也可继承YH—1软件财富并满足广大用户的要求。

1 语言扩充的实现方法

在一个语言中增加新的语法单位，我们说是对这种语言的扩充。按编译程序原有的构思和处理方法从源代码的接受到目标代码的生成均增加对新的语法单位的处理，我们称之为常规处理方法，否则称为非常规方法。按常规方法扩充编译器，可以维持其本身结构原貌，但一般要涉及整个编译程序的改动，修改和调试都比较困难。最常用的非常规方法是进行从源程序到源程序的转换[3]，相当于在编译前端增加了一个源程序预处理，其优点是活动余地大，可以做复杂的程序变换，但缺点是开销太大[4]。因此人们想了一个折衷的方法，即嵌入式的结构方法[4][5]，它将接受新的语法单位和转换源程序的工作放到编译程序的第一遍扫描(pass 1)中完成。

我们希望尽量按常规方法扩充YFT77[2]。但是按常规方法在YFT77中扩充向量语句将会遇到许多困难。主要是：

(1) 表达式处理工作量大，难度高(PASS1)。

由于增加了新的语法单位（三元符数组、向量符数组、向量函数调用、数组表达式等），整个表达式处理程序[6]都要有较大的改动。若另编一套向量表达式处理程序，意

1990年1月16日收稿

味着重编3000多行汇编程序，并且向量表达式处理程序与原有的表达式处理程序要有很多接口（因为向量表达式中也有标量表达式），调试非常困难。

（2）难以组织向量运算（第二遍PASS 2）

PASS 2生成目标代码的主要依据是中间代码。由于中间代码做了改动，势必导致PASS 2从基本块划分开始至目标代码生成的所有程序的全面更改。特别是根据向量语句的中间代码组织向量循环，循环的次数、向量访存首地计算、步长等还要回头查询PASS 1的表格，并且组织循环要产生标号到PASS 2已非常困难（因为所有标号在PASS 1结束时已经拉链完毕）。

因此，我们采用折衷的非常规方法，在PASS 1中将向量语句（组）源代码处理成可向量化的标量DO循环的中间代码，使得向量语句对PASS 2透明。对PASS 1修改后的流程如图1所示：

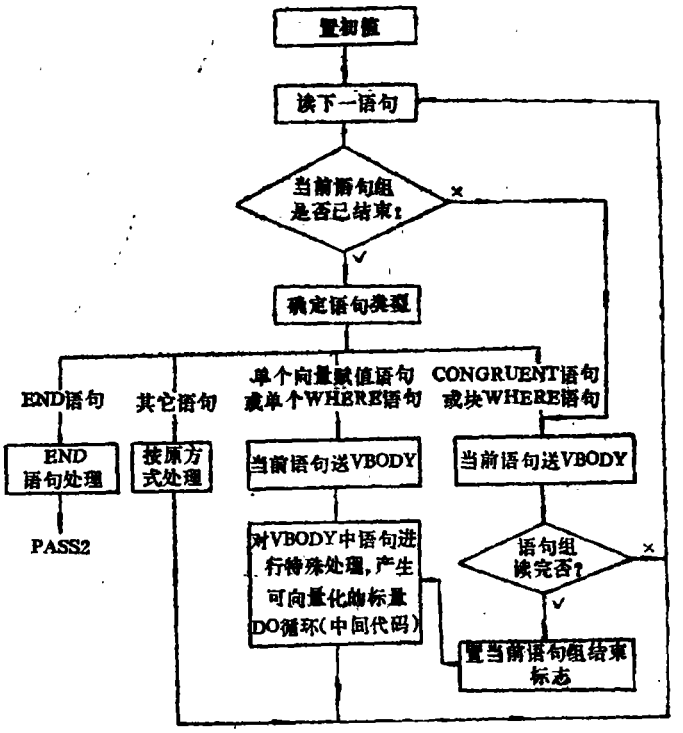

图 1 加向量语句后的PASS 1框图

2 主要技术难点及解决方法

将向量语句（组）转换成语义等价的可向量化DO循环，我们解决的主要技术难点如下：

2.1 DO循环的构造

一个向量语句（组）中的所有数组操作数都是相容的，因而可用一个等价的DO循环表示它。我们采用标准的DO循环形式[7]，即循环的三个参数为$E_1=1$，$E_2=M$，$E_3=1$．终值M是这样确定的：任取该向量语句（组）的某个数组操作数，若该数组操

作数以三元符数组形式出现，如：A(…，e_1:e_2:e_3，…)，则对应维循环终值E_2应满足：

$$M=\lfloor(e_2-e_1)/e_3\rfloor+1 \tag{1}$$

若该数组操作数以一般数组形式A出现，那么M则要通过查数组描述表而确定。

我们所处理的向量语句中，所有数组下标都是线性变化的，因而可认定A(…,e_1:e_2:e_3，…)对应的标量形式为A(…，aI+b,…)，而a，b应满足：

$$I=1：a+b=e_1 \tag{2}$$

$$I=M：a*M+b=e_2 \tag{3}$$

由(1)(2)(3)式可解得：$a=e_3$，$b=e_1-e_3$

故DO循环的最终形式为：

```
      DO α I1=1, ⌊(e12-e11)/e13⌋+1
                 ⋮
      DO α In=1, ⌊(en2-en1)/en3⌋+1
         A((I1-1)*e13+e11, …, (In-1)*en3+en1)=…
   α CONTINUE
```

此外，DO循环的终结标号、控制变量是由编译系统内部产生的，向量语句中带$号的临时数组的处理也采取了相应的措施。总之，避免了与用户程序中的变量与标号重复。

2.2 数据相关性问题的处理

本文所提方案的实现实质上是一种程序变换。为保证程序变换的正确性，必须讨论数据相关性问题。

例如，对于向量语句组

```
      CONGRUENT 2
      A(2:N-1)=B(2:N-1)+C(2:N-1)
      D(2:N-1)=A(1:N-2)*A(3:N)
```

若不考虑数据相关性，而将其转换为不可向量化的标量DO循环：

```
      DO α I=2, N-1
      A (I) =B(I)+C(I)
   α  D(I)=A(I-1)+A(I+1)
```

显然，转换前后运行结果不一致，这种变换是错误的。

在程序变换过程中必须先分析数据相关性是具有普遍意义的，但我们在具体实现时采取了一种更简便、实用的方法而避免了复杂的数据依赖关系分析。YFT77提供有编译指导命令CDIR$ IVDEP[9]，只要在直接变换所得的DO循环前加上这条命令就可保证系统将无条件地忽视影响向量化的数据相关性而对该DO循环进行向量化，产生与原向量语句等价的向量目标代码。

2.3 带条件向量语句的处理

由于YFT77不能对IF语句进行向量化，所以在处理带条件的向量语句（即条件

CONGRUENT语句和WHERE语句）时，也不希望产生IF语句，而是利用了YFT77提供的CVMGT函数[9]。

例如：

```
WHERE (A(1:N), GT.B (1:N))
  A (1:N)=B(1:N)
  B (1:N)=D
  ENDWHERE
```

将变换为：

```
      DO  α I=1, N
      A(I)=CVMGT (B(I), A(I), A(I). GT. B(I))
α     B(I)=CVMGT (D., B(I), A(I). GT.B(I)).
```

3 结 语

我们目前能够处理的向量语句有：基本数组赋值语句，连续数组赋值语句，一致性数组赋值语句，WHERE语句，块WHERE语句，CONGRUENT语句，CONGRUENT 语句组并允许I/O语句中出现三元符数组。只有PACK、UNPACK 语句没有实现。这已能满足绝大多数用户向量程序的要求。YFT77向量扩充版本已在YH—1系统上运行，并已交付使用，深受用户欢迎。

本文所提出的设计思想也适用于一般意义的向量程序标量化，因而对于向量软件的移值及向量程序本身正确性的验证也是极有意义的[8]。

这项工作得到杨桃栏、李兆盛副教授的热情帮助和汪诗林、韦海亮、赵克佳同志的大力支持，在此表示感谢。

参 考 文 献

[1] Cray Copy. CFT Internal Reference Mannal

[2] 赵克佳.技术报告：YFT字符扩充版实现方案.1988, 11

[3] Williams Harrison. An Overview of the Structure of the Parafrase, July 9, 1985

[4] 张可军. 向量识别器结构设计，国防科大研究所硕士论文, 1987, 1

[5] 吴少岩，郭克榕，张可军. 嵌入式向量识别器. 计算机工程与科学, 1987, 1

[6] YFT77源程序. 国防科大, 1989, 8

[7] Bruce Leasure. The Parafrase Project's Fortran Analyzer Major Module Documentation. Jtly 9, 1985. 29

[8] Michael Wolfe, Vtpal Banerjee. Data Dependence for Parallelism Detection. 1987

Implementation of Vector Statements in YFT77

Zhang Kejun Guo Kerong
(Department of Computer Science)

Abstract

A language extension method through the transformation from source language to intermediate language is presented in this paper. This method has been used in the implementation of vector statements in YFT77 and it has been proved to be efficient and practical.

Key words programming language, source program, vector statement/intermediate language, vectorization, language extension, program transformation