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

并行程序设计语言在串行机上的编译实现

郭　强
（成都军区）

摘　要　目前，诸如Fortran 8x程序设计语言中已经引入了并行运算成份。这给普通串行（标量）机上实现编译器带来了困难。本文提出了数组（矩阵）运算串行化的概念，并给出了串行化的判别准则及算法。

关键词　并行程序设计，矩阵，串行机/数组（矩阵）运算，串行化，原形串行化
分类号　TP314

随着向量流水线、多处理机大型系统的研制和开发，计算机高级语言中相应的并行运算语言成份应运而生。Fortran 8x[2]就是含有数组运算成份的程序设计语言。“银河”计算机Fortran编译器[3]具备了多种数组运算处理功能，除四则运算外，还能处理数组的逻辑运算及逻辑控制等。现在，用这种并行Fortran编写的大型应用程序也有不少，这些程序只能在具有并行处理机制以及并行语言编译器的系统上运行。然而，目前正在运行的计算机系统中绝大多数还是串行机制，甚至还没有一个串行的普通计算机上实现了Fortran 8x编译。本文探讨在普通串行（标量）机上如何实现数组运算。

我们讨论数组运算的串行计算的目的是为了在普通串行（标量）机上开发并行程序设计语言的编译器，这是属于编译方法的题目。在串行机制下，数组语句运算通常是由等价的DO循环来实现的。但数组语句在变换成为DO循环时存在所谓的“不一致性”问题。请看下面的例子：

例1　将数组运算语句直接变换成DO循环

$$A(1:50)=A(50:1:-1)+A(1:50)$$

变换后：

```
      DO ⓐ I=1,50
ⓐ     A(I)=A(51-I)+A(I)
```

将循环展开成下面的执行序列：

```
A(1)=A(50)+A(1)
.....
A(25)=A(26)+A(25)
.....
```

1989年11月11日收稿

$$A(50)=A(1)+A(50)$$

以上序列并行执行时，从第25句到第50句中引用的数组元素均为变更（赋值）前的值。但在串行执行时，即DO ⓐ循环执行时，引用的这些数组元素均为变更后的值。因此导致计算结果的不一致性。

由此可以看出，数组运算语句并非都可直接变换成串行计算格式，重要的是变换的等价性。为了确保数组运算与串行DO循环的计算结果一致，在变换前有必要对数据依赖关系[6]进行分析。下面各节分别讨论不同类型的数组运算语句的串行计算。由于描述数组运算的并行语言尚未有正式的标准文体，故本文以实现了的银河机的YHFT为语言参考“文本”[3]。

在并行Fortran中，参加运算的操作数不仅可以是变量，数组元素，常数等标量成份，而且允许数组作为操作数。数组的一部分或全部元素参与运算时，有一种简单的表示方法，这就是“三元符数组”[3]。例如，例1中的数组运算，它很象DO循环的一个缩写，表示出数组运算的上界、下界和跨度。在本文的讨论中均采用这种表示方法。

1　数组赋值语句的串行化

数组运算包括赋值运算，逻辑控制等，用得最广泛的是数组赋值语句，故我们首先讨论赋值语句的串行化问题。所谓数组赋值就是一个数组表达式的值或一个标量表达式的值赋给一个数组变量。形式描述如下：

<aiiay variable> = <expression>

在串行机制下，如果参与运算的数组元素可在编译时确定的话，那么，将数组运算展开成等价的串行语句序列，就可达到串行计算的目的。但是，并非所有的数组运算都能够在编译时确定参与运算的元素，因此，这种方法是不现实的，而且用展开方式实现数组运算会使程序变得雍长不堪。数组运算语句通常是用等价的DO循环来实现。

定义1　串行化是指将数组运算语句变换成等效的串行计算格式的过程。

定义2　原形串行化是将数组赋值语句的三元符下标替换成含循环控制变量的下标表达式，在该语句前增设一个DO语句，且以此赋值语句为循环终结句，如果形成的DO循环串行计算格式与原数组赋值语句等效（计算结果一致），则称为可原形串行化。

例2　可原形串行化的数组运算

$$A(M:N:S)=B(M1:N1:S1)+C(M2:N2:S2)$$

串行计算格式：

$$DO\ ⓐ\ I=0,\left\lfloor\frac{N-M}{S}\right\rfloor$$

$$ⓐ\quad A(S*I+M)=B(S1*I+M1)+C(S2*I+M2)$$

可原形串行化的意义就在于不需要引入临时变量（除DO变量之外）或临时数组而直接可以串行化。什么样的数组运算语句可原形串行化呢？下面给出：

判别准则1

在数组赋值语句中，如果被引用的数组不与左部被定值的数组同名，或不通过Fortran的等价(EQUIVALENCE)、公用(COMMON)语句，以及哑实结合产生间接同

名，则原数组赋值语句可原形串行化。

例2是最简单的数组赋值语句，数组A与数组B、C不同名，也就不会发生同名数组的依赖关系，因此，可直接转化成串行计算格式，即可原形串行化。

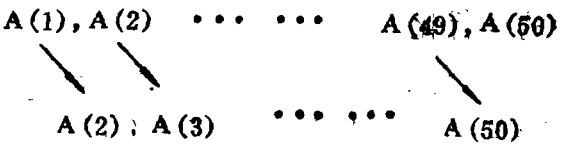
所谓通过等价、公用语句间接同名，是指程序设计中采用等价或公用语句结合共享存贮空间而造成的数组或变量之间的间接等价性；哑实结合同样可以使子程序段中不同名的数组或变量之间具有相同的存贮单元。文[5]已深入讨论了这个问题。

例1的数组赋值语句中，左部被定值的数组与右部被引用的数组同名，这就不能直接串行化，要进行数据依赖关系的分析[6]。这里所说的“数组同名”隐含了前述的间接同名，以下同样。

判别准则2

一个数组赋值句可原形串行化的充分条件是该语句中引用性的数组出现不依赖于被定值的同名数组。

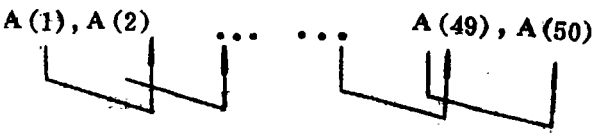
例3 数组赋值语句A(2:50)=A(1:49)在并行计算机上执行的方式如下：



而下面的循环

```
      DO 3 I=2,50
3     A(I)=A(I-1)
```

的执行方式是：



在串行格式中，A的每个元素的赋值均依赖于前一个元素的值；并行计算时，赋值用的都是数组A的“旧值”，故计算结果不一致。这是由于A(I-1)依赖于A(I)的缘故。因此，该数组赋值语句的串行化可引入临时变量或数组把“旧值”暂存起来，以便下次循环时赋值。该数组赋值语句等价的串行格式为：

```
      T1=A(1)
      DO 3 I=2,50
      T2=A(I)
      A(I)=T1
3     T1=T2
```

其中T1,T2是引入的临时变量，该例既可采用改变增量为负值的方式实现串行格式：

```
      DO 3 I=49,1,-1
```

```
      3  A(I+1)=A(I)
```

例4　（例1的继续）

在数组赋值语句中

$$A(1:50)=A(50:1:-1)+A(1:50)$$

A(1:50)与A(50:1:-1)存在多对相关点[6]：<1,50>、<2,49>、…，在执行区间[1,25]上是交叉依赖的，不能满足判别准则2的条件，故该数组赋值语句不能直接化为串行格式，需通过临时暂存数据，才能串行化。串行格式如下：

```
      DO 4 I=1,25
      T1=A(I)
      A(I)=A(51-I)+A(I)
   4  A(51-I)=T1+A(51-I)
```

一般而言，每个数组赋值语句必定能够找到与其等效的串行计算格式。譬如，用DO循环代替数组赋值句时，引入足够多的临时数组，如例3，把需要引用的数组元素变更前的值（旧值）暂存在新开辟的临时数组中，而后在串行DO循环中，在引用原数组的地方引用该临时数组。

数组赋值语句串行化算法

(1) 检查数组赋值语句的依赖关系，不妨设数组赋值语句的一般形式为：

$$A(M:N:S)=A_1(M_1:N_1:S_1)+A_2(M_2:N_2:S_2)+\cdots$$

其中$A, A_1, A_2, \cdots$分别是数组名，某些甚至是同名的数组。

(2) 若上式符合判别准则1或2，则将其原形串行化，结束；

(3) 若上式不符合判别准则1或2，则必存在$i_1, i_2, \cdots$，使得A与$A_{i1}, A_{i2}, \cdots$同名，定义若干个临时数组$T_1, T_2, \cdots$，且与$A_{i1}, A_{i2}, \cdots$的维长对应相等；

(4) 在原数组赋值语句前产生若干新的暂存赋值语句：

$$T_1(M_{i1}:N_{i1}:S_{i1})=A_{i1}(M_{i1}:N_{i1}:S_{i1})$$
$$T_2(M_{i2}:N_{i2}:S_{i2})=A_{i2}(M_{i2}:N_{i2}:S_{i2})$$
$$\cdots\cdots$$

(5) 改写原数组赋值语句，其中对$A_i(i=i_1, i_2, \cdots)$的引用改为对$T_j(j=1,2,\cdots)$的引用，即：

$$A=(M:N:S)=\cdots+T_1(M_{i1}:N_{i1}:S_{i1})+\cdots$$

(6) 将新产生的数组赋值语句序列及改写后的数组赋值语句分别原形串行化；

(7) 算法结束。

串行计算格式不是唯一的，在实际工程中应考虑优化结构。

上面讨论的是一维数组串行化的情形。一般而言，数组的维数就是变换后DO循环嵌套层数。多维数组赋值语句串行化的算法与以上给出的算法完全一致，只是数组下标的描述更加复杂。这里就不再赘述。

2　数组赋值语句序列的串行化

前面已经讨论了单个赋值语句的串行化问题。对于多个赋值语句序列串行化有两种

方法：其一，按单个赋值语句的处理方式，分别组织DO循环，然后再将其中一些可合并的循环合并，我们称之为“局部——合并”串行化；其二，是用本节所给的判别准则的“整体串行化方法”。

2.1 “局部—合并”串行化算法

先运用第1节中给出的串行化判别准则将各赋值语句分别串行化，令串行化后的串行DO循环序列为：$D_1, D_2, D_3, \cdots\cdots D_n$. 如果 D_i 与 $D_{i+1}(i=1,2,\cdots\cdots n)$ 长度（指循环次数）相等，且无相关的数组[6]出现在D_i与D_{i+1}中，则D_i可与D_{i+1}合并。若 D_i 与 D_{i+1} 中有相关的数组出现，则要分析依赖关系，例如 D_i 中的数组出现依赖于 D_{i+1} 中的数组出现，则两者可以合并；反之则不能合并。

有时 D_i，D_{i+1} 及 D_{i+2} 中的 D_{i+1} 不能与 D_i 或 D_{i+2} 合并，为了使 D_i 与 D_{i+2} 合并就需要将 D_i 与 D_{i+1} 交换，其交换的条件是 D_i 与 D_{i+1} 中参与运算的数组不相关。

2.2 整体串行化算法

多个数组赋值语句之间可能存在数据依赖关系，即有相关数组存在。为了确保串行化的正确性，应当进行数据依赖关系分析，文[6]中的定义和本文一致，故不再赘述。

判别准则3

多个数组赋值语句序列可同时整体串行化的必要条件是：

(b) 循环长度相等；

(a) 单句均分别满足判别准则1或2；

(c) 数组赋值语句之间无依赖关系。

3 结 束 语

由于篇幅所限，条件数组赋值的控制结构型语句的串行化问题将在以后的文章中讨论。另外，并行FORTRAN中还引入了向量或数组内部函数、语句函数，在串行化过程中应作相应的变换。向量外部函数是由用户自己定义的，在串行化过程中也应特别留意。

文中提出的几种判别准则都是充分条件，只有满足条件才能变换成等价的DO循环，而且尚未考虑优化方法。

参 考 文 献

[1] 张里航.新的语言建议标准FORTRAN 8x.计算机世界报，1989,(41)

[2] American National Standards Institute, Proposals Approved for Fortran, 8x, X212/8,80 (preliminary document), Inc, 1981,(30)

[3] 樊哲民.程序变换概论.电子学报，1983，(1)

[4] 范植华，郭强，刘会敏.向量化中隐数据依赖关系的分析.计算机学报，1985，(5)

[5] 郭强，陈海波.串行运算向量化的相关分析方法.计算机学报，1985,(5)

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

正切函数的契比协夫逼近及其系数的加速计算法

李晓梅 王宏斌 赵自春
(计算机系)

摘 要 本文给出正切函数的有理展开和契比协夫展开式系数的加速计算方法；同时在YH-1机上进行了数值试验，结果表明：用现在的系数来计算正切函数，其精度比原来正切函数的精度可提高30%左右。

关键词 逼近法，函数，绝对误差，相对误差/数值计算，契比协夫多项式
分类号 O241.8

在进行正切函数数学子程序方案设计时，首先利用它的有理展开式来构造契比协夫逼近式，再使用契比协夫逼近式构造有理分式逼近式及进行误差估计。因此，本部分

1989年7月28日收稿

The Compiler Implementation of Parallel Programming Language on Serial Machine

Guo Qiang
(PLA Chengdu Military-Area)

Abstract

At present the array-array (matrix-matrix) opreation has been introduced into parallel programming languages, such as Fortran 8x. This makes compiler implementation on serial machine difficut. This paper proposes the concept of "serialization" and gives the algorithm about serial opreating.

Keg words parallel programming, matrix, serial computer/array-array (matrix-matrix) opreation, serialization, original shape serialization