国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第10卷第1期 1988年3月 Vol.10 No.1

FORTRAN 程序的多任务分割

杨学军 陈立杰
（计算机系）

摘 要 本文以数据相关分析为基础，详细地讨论了LOOP级多任务生成的几种方法，给出了FORTRAN循环可并行执行的条件，提出了需要注意的几个问题。在此基础上，本文提出了对FORTRAN程序进行宏任务生成的一种新方法，并且考虑了实现多任务生成的几个编译问题。

关键词 FORTRAN程序，多任务分割

1. 引 言

随着计算机系统的广泛应用，对计算机运算速度的要求越来越高。目前，只靠提高器件的速度指标已经远远不能满足这种要求，因此，研究计算机的并行体系结构将成为今后计算机的发展方向。多机系统具有灵活、可靠、高速三大优点，在当今的计算机市场上已显示出其优势。

为了有效地利用多机系统，需要把一个实际问题划分成多个并发任务。一般情况实现多任务划分有两种方法：一种是算法设计，一种是自动识别。设计适用于多机系统的算法要求程序员对多机系统有较深刻的理解，这种负担对非计算机专业的用户来说是非常沉重的。所以研究多任务的自动识别技术无论在理论上还是在实践中都有重要意义。

本文旨在探讨FORTRAN程序的多任务自动生成技术。

2. 基 本 概 念

1) Bernstein 条件

为了解决串行程序并行化问题，Bernstein提出了一组基本条件：

假设有程序段（如图1）。

```
  ⋮
 s1
 s2
  ⋮
```

图 1

如果下列条件成立：

(1) $IN(s1)\cap OUT(s2)=\Phi$；(2) $IN(s2)\cap OUT(s1)=\Phi$；

(3) $OUT(s1)\cap OUT(s2)=\Phi$

则语句s1，s2可并行执行，其中：

1987年7月1日收到

IN(S)：表示语句S的读引用变量的集合；

OUT(S)：表示语句S的写引用变量的集合。

2） 数据相关

用 Bernstein 条件来分析程序需要对每个语句S求I(S)和 OUT(S)，并且对其求相交运算，这是一件很麻烦的事。本文试图以数据相关分析为手段，对程序的并发特性进行探讨。下面给出几个定义。

（1）流相关：如果存在一条从s1到 s2 的执行路径并且存在一个被 s1 赋值又被 s2 引用的变量，则称 s1 和 s2 有流相关关系。记作：$s1 \longrightarrow s2$

（2）反相关：如果存在一条从 s1到 s2 的执行路径并且存在一个被 s1 引用又被 s2 赋值的变量，则称s1和s2有反相关关系。记作：$s1 \not\longrightarrow s2$。

（3）写相关：如果存在一条从s1到 s2 的执行路径并且存在一个被 s1 赋值又被 s2赋值的变量，则称s1和s2有写相关关系。记作：$s1 \circ\!\!\!\!\!\!\longrightarrow s2$。

（4）数据相关流图：如果图$G=\langle V, E\rangle$满足如下条件：

（a）V是由程序段s 的所有语句构成的集合。

（b）$E=\{(s1, s2) \mid (s1 \longrightarrow s2 \vee s1 \not\longrightarrow s2 \vee s1 \circ\!\!\!\!\!\!\longrightarrow s2)\}$

则称图G是程序段S的数据相关流图。

以上给出了数据相关关系的定义，不难看出Bernstein条件和数据相关关系存在着内在的联系。

结论 1

如果语句s1和s2没有数据相关关系，当且仅当它们满足Bernstein条件。

3） 基本并发机制

为了在本文中清楚地表示并发性，假定系统提供了如下几种并发机制：

- TSKSTART(ID)：创建名为 ID 的任务。
- TSKWAIT(ID)：等待名为ID的任务完成。
- EVPOST(EV)：发送事件。
- EVWAIT(EV)：等待事件发生。
- EVCLEAR(EV)：清除事件。
- LOCKON(LOCK)：上锁。
- LOCKOFF(LOCK)：开锁。

3. Do 循环的纵向分割

有人做过统计、一个程序的 90% 的运行时间花费在循环部分。所以 研究 循环的并行性，在实际应用中起着举足轻重的作用。本文以Do循环为突破口，研究FORTRAN程序的并行执行问题。

1） Do 循环的异步并发性

例 1 假如有如下FORTRAN循环。

```
Do  10  I=2, N
  A(I)=A(I-1)+C          s1
```

```
      B(I)=B(I-1)*D          s2
   10 CONTINUE
```

不难看出此循环可以改写成如下两个可异步执行的独立任务。

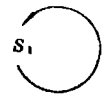
```
      TASK 0
     Do  10  I=2, N
        A(I)=A(I-1)+C
   10 CONTINUE
      TASK  1
     Do  10  I=2, N
     B(I)=B(I-1)*D
   10 CONTINUE
```

对于一般情况，可导出如下结论。

结论 2

如果一个循环体的相关关系图含有 n 个分图，则此循环可以表示成 n 个任务。

对于例1，循环的相关关系图见图2。



显然它是一个二分图，所以原循环可以改写成二个任务。

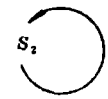


图 2

2） **Do 循环的同步并发性**

当然并不是所有的 Do 循环均符合上述条件，找出这方面的反例是很容易的。

例 2

```
      Do  10  I=2, N
       A(I)=C(I)+D          s1
       B(I)=A(I-1)          s2
     10 CONTINUE
```

因为它的相关关系图如图3：

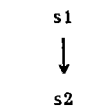


图 3

所以不能用上述方法将其改写成多任务。但是如果在循环尾部引入同步原语情况就不一样了（称这种方法为尾同步方法）。

```
      TASK  0
      Do  10  I=2, N
      A(I)=C(I)+D
      CALL  EVPOST(E1)
   10 CONTINUE
      TASK  1
      Do  10  I=2, N
      CALL  EVWAIT(E1)
      B(I)=A(I-1)
      CALL  EVCLEAR(E1)
   10 CONTINUE
```

为了叙述方便，首先引入如下定义。

定义 假设语句 s1 和 s2有相关关系，即：$s1 \longrightarrow s2 \vee s1 \not\rightarrow s2 \vee s1 \overset{0}{\rightarrow} s2$，且数组元素 A(f(I))和A(g(I))分别在 s1和 s2中出现，即：

```
      Do  1  I=1, N
          ⋮
      ··· A(f(I)) ···     s1
          ⋮
      ··· A(g(I)) ···     s2
          ⋮
    1 CONTINUE
```

(1) 如果存在I_1和I_2，使等式$f(I_1)=g(I_2)$成立，并且I1<I2，则称 s1和 s2 有正相关关系。记作：$s1 \overset{+}{\rightarrow} s2$。

(2) 如果存在 I_1，使等式 $f(I_1)=g(I_1)$成立，则称 s1 和s2有 0 相关 关系。记 作：$s1 \overset{0}{\rightarrow} s2$。

(3) 如果存在I_1和I_2，使等式$f(I_1)=f(I_2)$成立并且I1>I2，则称s1和s2有负相关关系，记作：$s1 \overset{-}{\rightarrow} s2$。

对于例 2，语句s1和s2有正相关关系。（图 4）。

必须注意的是，有时两个语句既具有正相关又具有 0 相关和负相关。

定义 如果图$G_0=\langle V, 0.\rangle$满足如下条件：

(1) V是程序段s 的所有语句构成的集合；

(2) $E_0=\{(s1, s2) \mid s1 \overset{0}{\rightarrow} s2\}$

则称G_0为程序段 s的 0 ——相关关系图。

例 2 的 0 ——相关关系图是图 5 。

此例帮助我们揭示了如下一条规律。

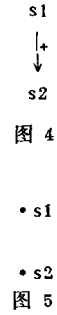


图 4

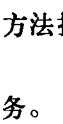


图 5

结论 3 如果一个循环体的 0 ——相关关系图有 n 个分图，则可用尾同步方法把此循环分割成多任务。

除了尾同步方法之外，我们可以用其它一些同步方法把此循环表示成多任务。

例 3

```
      Do  10  I=1, N
        A(I)=2*I
        B(I)=2+I
        C(I)=A(I)+B(I)
   10 CONTINUE
```

可以变换成如下两个任务

```
      TASK  0
      Do  10  I=1, N
        A(I)=2*I
        EVWAIT(E1)
```

```
      EVCLEAR(E1)
10    CONTINUE
        TASK1
      Do  10  I=1, N
        B(I)=2+I
        EVPOST(E1)
10    CONTINUE
```

实际上，任何一个程序段（语句数多于2），通过插入适当的同步原语，均可以表示成多任务。但是，在最坏的情况下这些任务的执行还是串行的。既便是一个程序段中有一些可以并行执行的成份，在引入同步原语时也要十分小心。因为在实际系统中同步原语的开销不容忽略。过多地引入同步原语，不但不能改进程序的执行，反而会降低程序的运行效率。从另一个方面来说，我们希望同步原语的开销要尽量地小。要做到这一点，要求系统程序员必须认真地考虑所采用的算法，并要求硬件提供很强的支持。

4. Do循环的横向分割

上一节讨论了循环的纵向并发性，不难看出其条件是相当苛刻的。虽然引入同步原语可以适当地放宽这些条件，正如上一节所述，过多地引入同步原语显然是有害无益的。这就促使我们寻找新的方法，来解决Do循环的并行执行问题。

1) Do循环的横向并发性

首先通过一个例子来观察Do循环的横向并发性。

例 4

```
      Do  10  I=1, 2N
        A(I)=C(I)+B(I)
        E(I)=A(I)+F(I)
10    CONTINUE
```

可以改写成如下两个独立的子任务：

```
        TASK  0
      Do  10  I=1, N
        A(I)=C(I)+B(I)
        E(I)=A(I)+F(I)
10    CONTINUE
        TASK  1
      Do  10  I=N+1, 2N
        A(I)=C(I)+B(I)
        E(I)=A(I)+F(I)
10    CONTINUE
```

对于一般情况有如下结论。

结论 4 如果一个循环体每两条语句之间均没有正相关关系或负相关关系，那么此循环可变换成多任务。

对于一个循环来说，横向分割方法是很有实用价值的。如果循环长度为N，可以将其变换成n个任务($1 \leqslant n \leqslant N$)。例4给出了$n=2$的变换方法。当n∤N时稍有麻烦，只要将剩余部分单独作为一个任务，就迎刃而解了。至于n选多大合适，这要据目标机所含处理机的个数而定。

当然，结论4只给出了充分条件，不满足这个条件也可以将一些循环划分成多任务。

例 5

```
      Do  10  I=3, 2N
        A(I)=A(I-2)+C
   10 CONTINUE
```

可以改写成如下二个独立任务：

```
        TASK 0
      Do  10  I=3, 2N-1, 2
        A(I)=A(I-2)+C
   10 CONTINUE
        TASK 1
      Do  10  I=4, 2N, 2
        A(I)=A(I-2)+C
   10 CONTINUE
```

这类问题和相关点的距离有关系，在此处不作更深入的讨论。

2） 标量循环的处理

如果一个循环中含有标量变元，那么直接运用横向方法分割Do循环是危险的。

例 6

```
      Do  10  I=1, N
        A=C(I)+D(I)
        B(I)=A+2
   10 CONTINUE
```

在改造此循环时，必须考虑存贮冲突问题，因为这时A是多个任务的共享数据。为了解决这一问题，可采用标量扩张技术。

例6的循环可以改造成：

```
      Do  10  I=1, N
        #A(I)=C(I)+D(I)
        B(I)=#A(I)+2
   10 CONTINUE
```

还有一类问题，不能运用标量扩张技术，这时可以采用加临界区的办法。

例 7

```
      Do  10  I=1, 2N
        S=S+A(I)
   10 CONTINUE
```

可以改写成如下两任务：

```
      TASK 0
    Do  10  I=1, N
      LOCK ON (L)
      S=S+A(I)
      LOCK OFF (L)
10  CONTINUE
      TASK 1
    Do  10  I=N+1, 2N
      LOCK  ON (L)
       S=S+A(I)
      LOCK  OFF (L)
10  CONTINUE
```

用横向方法开发循环的并发性是一种行之有效的方法，它便于实现、识别率高。是FORTRAN程序并行执行的重要手段之一。

5. 任务的合并

为了支持用户级多任务的开发，需要系统提供足够的原语。包括：任务创建，任务同步，任务互斥等等。调用这些原语的时间开销很大，更值得注意的是，这种开销随着任务数的增加而增加。所以，在实践中，任务数量越多并不一定越好。

1) 小任务的合并

鉴于以上的分析，合并一些任务特别是小任务是很有意义的。本文下面将提出一种新的任务合并技术。假设有任务T1，T2，T3，T4，T5、T6，它们之间的关系（即：优先关系）如下图6所示：

实际上T2和T3并不能并行执行，所以合并这两个任务对提高运行效率是有好处的，合并之后的优先关系图如图7所示：

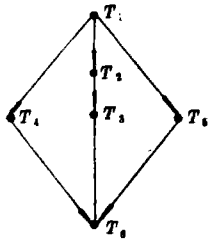


图 6

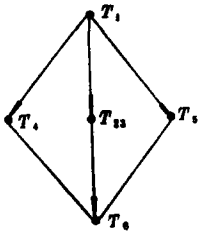


图 7

即使是可以并行执行的任务，在处理机受限的情况下，也可以考虑采用这种策略。

下面还是以具体的例子来揭示任务合并必要性与可行性。

例 8

```
      Do  10  I=1, 2N
        A(I)=B(I)+C(I)
10    CONTINUE
```

```
      Do  20  I=1, 2L
        D(I)=2*A(I)
   20 CONTINUE
```

按前几节介绍的方法，这段程序可以分割成四个小任务：

```
      TASK 0                          TASK 1
    Do  10  I=1, N                  Do  10  I=N+1, 2N
      A(I)=B(I)+C(I)                  A(I)=B(I)+C(I)
 10 CONTINUE                     10 CONTINUE
      TASK 2                          TASK 3
    Do  20  I=1, L                  Do  20  I=L+1, 2L
      D(I)=2*A(I)                     D(I)=2*A(I)
 20 CONTINUE                     20 CONTINUE
```

我们可以把 4 个任务合并成两个任务：

```
      TASK 02                         TASK 13
    Do 10  I=1, N                   Do  10  I=N+1, 2N
      A(I)=B(I)+C(I)                  A(I)=B(I)+C(I)
 10 CONTINUE                     10 CONTINUE
        同    步                          同    步
    Do  20  I=1, L                  Do  20  I=L+1, 2L
      D(I)=2*A(I)                     D(I)=2*A(I)
 20 CONTINUE                     20 CONTINUE
```

2） 减少同步

如上例所示，为了合并小任务，我们在每一个循环之外引入了一次同步。虽然和任务创建、撤消原语相比，同步原语的开销要小得多，特别是对于提供了硬同步机制的机器情况更是如此，但是删除多余的同步调用，仍有十分重要的意义。

例 9

```
      Do  10  I=1, 2N
   10 A(I)=B(I)+C(I)
      Do  20  I=1, 2L
   20 D(I)=2*B(I)
```

可以变换成如下两个任务

```
      TASK 0                          TASK 1
      Do  10  I=1, N                  Do  10  I=1+N, 2N
 10 A(I)=B(I)+C(I)               10 A(I)=B(I)+C(I)
      Do  20  I=1, L                  Do  20  I=L+1, 2L
 20 D(I)=2*B(I)                  20 D(I)=2*B(I)
```

3） 控制语句的处理

以上只讨论了一些简单的情况，对于一些复杂问题，必须考虑程序的控制流程。

例 10

```
      PROGRAM  EXAMPLE
      Do  10  I=1, 2*N
10    A(I)=B(I)+C(I)
      IF (Y.EQ.0.0) Go  To  30
      Do  20  I=1, 2*L
20    D(I)=2*B(I)
30    Z=H+G
      M=L+N
      END
```

对于如上这段程序（为了方便，省略了说明和I/O语句），可以将其分割为两个任务：

```
      TASK 0
      PROGRAM  EXAMPLE
      TSKSTART(EX1)
      Do  10  I=1, N
10    A(I)=B(I)+C(I)
      IF(Y.EQ.0.0)Go  To  30
      Do  20  I=1, L
20    D(I)=2*B(I)
30    Z=H+G
      TSKWAIT(EX1)
      END
      TASK 1
      SUBROUTINE  EX1
      Do  10  I=N+1, 2N
10    A(I)=B(I)+C(I)
      IF (Y.EQ.0.0) Go  To  30
      Do  20  I=L+1, 2L
20    D(I)=2*B(I)
30    M=L+N
      END
```

不难看出，此例揭示了一条开发宏任务的道路，具体地说有三个步骤：

- 基本块上的多任务分割
- 控制流分析与小任务合并
- 减少同步调用

众所周知，FORTRAN程序的并行计算是一个难题，以上只是初步研究，后续工作正在进行。

参 考 文 献

[1] Kai Hwang: Advanced Parallel Processing and Supercomputer Architectures
[2] J.Larson: Multitasking on the Cray X—MP/2 Multiprocessor, IEEE Computer, pp. 62—29, July 1984
[3] Michael Joseph Wolfe: Optimizing Supercompilers for Supercomputers

Multitasking of Fortran Programs

Yang XueJun Chen LiJie

Abstract

Through data dependence analysis, this paper discusses the method of multitasking partition at the loop level, gives the condition of parallel execution of Fortran loops, and puts forward some questions needed to be solved. Finally, there will be the possibility of macro multitasking partition.

Key words Fortran Programs, Multitasking