国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第14卷第3期 1992年9月 Vol. 14 No. 3

含控制相关的循环并行执行

吴少岩 吴健安
（电子计算机系）

摘 要 文中定义了在当代向量计算机上可高效实现的 merge 与 pos 函数，提出利用二函数消去 do 循环中 forward－if 与 search－if 的方法。本文讨论了向量化 forward－loop 的一般性方法并引入封闭子图的概念，给出一种简化语句执行条件的途径。

关键词 向量计算机，循环，控制相关，约束条件，封闭子图

分类号 TP311．5

含控制相关的循环是指 do 循环体内包含 if 语句的循环，简记 if－loop．在超级计算机上，if－loop 的高效执行一直是系统设计者与编译设计者关注的问题。超级计算机的许多新的硬件特征，如 gather/scatter[1]、归并、递归等指令，可直接或间接地支持一些程序结构（如非线性数组引用[1]、if 语句、递归运算）的向量执行。然而，由于人们尚未充分认识到这些硬指令的有效性，因此，若循环中包含 if 语句，通常被认为是难以向量化的或只能串行执行的循环。

文中将循环体内的 if 语句按其转移的方式分为三类：（1）向前转移但不转出循环之外的 forward－if；（2）转出循环之外的 search－if；（3）向回转移但不转出循环之外的 backward－if．第三类 if 语句在循环体内形成非 do 循环[2]，此类循环的向量化取决于能否将非 do 转换为 do 循环。当循环体仅含(1)或仅含(2)类 if 语句时，分别称之为 forward－loop 和 search－loop．本文旨在讨论这两类循环的向量执行。

1 控制相关的分析与转换

程序中存在两类相关关系：数据相关与控制相关[3]。数据相关制约数据项之间的访问顺序，而控制相关则是程序中的控制流时序。在 if 语句分支上的语句其执行与否完全取决于 if 条件式的“真值”，称分支上的语句控制相关于 if 条件式。以语句为结点、弧表示相关方向的有向图是表示程序中相关性的有效方法[4]。用 $s_i\delta s_j$ 表示语句 s_j 数据相关于语句 s_i，$s_i\delta_c s_j$ 表示 s_j 控制相关于 s_i．在相关图上，用 $s_i \rightarrow s_j$ 和 $s_i \rightarrow s_j$ 分别表示同样的含义，如图1.

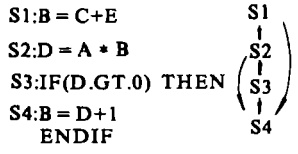


图1 含控制相关与数据相关的代码段与相关图

对数据相关的分析已有一整套基于数论与图论的分析算法[5]，我们也知道如何向量化仅含数据相关的循环[3]。然而，尽管在相关图上易于表示控制相关，但一个含控制相关

* 1990年6月20日收稿

的循环很难直接利用机器提供的向量硬件。因此，需开辟一条途径将控制相关转换为数据相关，变多分支的代码段为直级代码块。

“归并操作”是许多向量机具备的功能之一。归并指令 $(op_1, op_2, op_3) \rightarrow op_r$ 解释为：位串 op_3 第 i 位若为1，则结果操作数 op_r 的第 i 位具有 op_1 的第 i 位置的值；否则具有 op_2 第 i 位置的值。当归并指令为向量指令时，op_3 为向量屏蔽位串，而操位数的第 i 位即第 i 个向量元素。利用归并指令，易于设计出一个可向量化的归并函数：

$$\text{merge}(\text{exp1},\text{exp2},\text{lexp}) = \begin{cases} \text{exp1} & \text{lexp} = \text{'true'} \\ \text{exp2} & \text{lexp} = \text{'false'} \end{cases}$$

利用 merge 函数，图1可转换为图2.

这一转换仅仅是用 S_3' 与 S_4' 之间的数据相关取代了原代码段中 S_3 与 S_4 之间的控制相关，而未改变原程序的相关顺序。一般地，控制相关于条件 C 的赋值语句 x=exp 可以转换成 x=merge(exp,x,c). 但需指出，由于 exp 的计值在转换后不依赖于条件 C 的真值，若 exp 存在副作用，如被零除或调外部过程改变了 x 的值，则不能保证转换的等价性。当副作用不存在时，易于看出，转换前后在语义上等价并有下述结论。

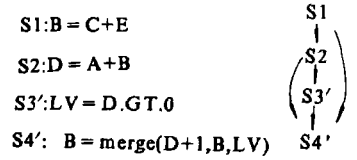


图2 转换后的代码段与相关图

定理1 代码段

$$\begin{array}{ll} C_1: & \text{if(cond) then} \\ & S_1 \\ & \vdots \\ & S_k \\ & \text{endif} \end{array} \qquad (1)$$

无论赋值语句序列 $S_1, \cdots, S_k$ 内部以及 cond 与 $S_1, \cdots S_k$ 之间有怎样的数据相关关系，总能转换成赋值语句序列：

$$\begin{array}{l} S_0': \text{lvar}=\text{cond} \\ S_1' \\ \vdots \\ S_k' \end{array}$$

这里，$S_j'(1 \leqslant j \leqslant k)$ 是 S_j 利用归并函数 merge 转换后的语句，lvar 是逻辑变量。（证略）

2 search—loop 的向量执行

对于含单个 search—if 的 search—loop：

$$\begin{array}{ll} & \text{do} \quad \alpha \quad I = 1,\text{iter} \\ & S_1 \\ & \vdots \\ & S_k \\ C: & \text{if(cond)} \quad \text{goto} \quad l \\ & S_{k+1} \\ & \vdots \\ & S_m \end{array}$$

α　　continue　　(2)

其中，l 在循环 do－α 之外定义，$S_i(1\leqslant i\leqslant m)$是赋值语句。定理1对 search－if 的转换无效。这是由于 search－if 语句 C 定义的语义为：在循环执行过程中，当 cond＝'false'时，语句 $S_1\cdots S_K$，$S_{k+1}\cdots S_m$ 顺序执行；而一旦 cond＝'true'，则循环体仅完成 $S_1\cdots S_k$ 的本次执行，控制转至 l 定义的代码位置执行。显然，不能根据 cond 的真值对赋值语句 $S_i(1\leqslant i\leqslant k)$作归并运算而消去语句 C.

定义　真值定位函数 pos

$$pos(larr,iter)=\begin{cases}n & 1\leqslant n\leqslant iter,\text{n 是逻辑数组 larr 中第一个真值为'true'的元素位置。}\\ iter+1 & \text{larr 的真值恒为'false'.}\end{cases}$$

其中，larr 是逻辑数组，iter 是循环迭代次数。

在许多向量机上可以找到 pos 函数很好的硬件支持。向量测试、标量位串数前导零指令直接支持这一函数的高效实现。实际上，引入 pos 函数的目的在于：该函数的值决定了 search－if 的动态迭代次数，因此，有可能利用该值取消循环中的 search－if. 例1是一个简单的 search－loop.

例1

```
      DO 5  I=1,100
      A(I)=(A(I)+B(I-1))/2.0
      B(I-1)=SIN(A(I))*2.0
      IF(A(I+1).GT.B(I))  GOTO 100
      C(I)=A(I)-A(I-1)
5     CONTINUE
```

如果 search－if 的条件式是循环不变式，则条件式的值恒为'true'(pos＝1)或恒为'false'(pos＝iter＋1). 这时不必计算 pos 函数。当条件式恒为“真”时，取消 search－if 及其后的循环体语句，循环蜕化为只完成第一次迭代的赋值语句序列并在该序列之后加无条件转换语句。当条件式恒为“假”时，则只需消去 search－if 语句本身。为叙述简便，下面的讨论总假定 search－if 的条件式是循环可变的。

定理2　(2) 式循环中的 search－if 是可利用 pos 函数消除的，当且仅当 C 与 G 之间在相关图上不存在任何回路。这里 C 是 search－if 语句，G 是赋值语句序列 S_1，…S_k，S_{k+1}，…，S_m 构成的数据相关子图。

证明　(1) 假定 C 与 G 在相关图上存在回路，这时相关图形如图3，在相关图中，C 数据相关于 G，即 C 中条件式 cond 的计值依赖于 G 内语句的计算；而 G 控制相关于 C，亦即 G 的下次执行又取决于 C 的计值结果，因而形成相关回路。由于这样的回路存在，则不能独立于赋值语句计算决定迭代次数的 pos 函数。故不能将语句 C 从循环体中消去。

图3

(2) 若 C 与 G 之间不存在回路，利用 pos 函数，可将(2) 重构成：

```
      do β I = 1,iter
β     larr(I) = cond
      n = pos(larr,iter)
```

```
        do γ I = 1,n - 1
           S1
           ⋮
           Sk
           Sk+1
           ⋮
γ          Sm
        if(n. EQ. iter + 1)goto l'
           S1^n
           S2^n
           ⋮
           Sk^n
        goto l
l':
```

这里，$S_i^n(i=1,\cdots,k)$代表语句 S_i 在执行第 n 次迭代时的实例。这样的重构保持了语义的等价性并保留了循环变量 I 应有的循环出口值。（证毕）

利用定理2，例1可重构成例2. 显然，例2中的 Do－10与 Do－20都是可向量化循环。

例2 （续例1）

```
      DO 10 I = 1,100
10    LARR(I) = A(I + 1). GT. B(I)
      N = pos(LARR,100)
      DO 20 I = 1,N - 1
      A(I) = (A(I) + B(I - 1))/2. 0
      B(I - 1) = SIN(A(I))*2. 0
20    C(I) = A(I) + A(I - 1)
      IF(N . EQ. 100 + 1)  GOTO  1000
      A(N) = (A(N) + B(N - 1))/2. 0
      B(N - 1) = SIN(A(N))*2. 0
      GOTO 100
1000
```

考虑一般的 search－loop：

```
        do   α  I = 1,iter
             G1
C1:     if(cond1)  goto l1
             ⋮
             Gp
Gp:     if(condp)  goto lp
             Gp+1
α       continue                                    (3)
```

其中，标号 $l_1,\cdots,l_p$ 均在循环之外定义；任意 $G_i(i=1,2,\cdots,p+1)$为包含若干个赋值语句的序列。从 pos 函数的定义容易得出下面的结论：

定理3 对循环(3)，在 C_1，…，C_p 与 G_1，…，G_p，G_{p+1}之间构成的相关图中，若任意 $C_i(1\leqslant i\leqslant p)$不处于相关回路中，则循环(3) 可以转换成循环(2)。

证明 根据命题设定的条件，循环(3) 的相关图如图4. 图中虚线框示意由 G_1，…，G_{p+1}构成的数据相关子图；图4忽略了对证明不重要的数据相关，如 $C_i\delta G_j$.

由于 C_1，…，C_p 之间不可能存在数据相关，因此，可以分别求每个 search－if 的 pos 函数：$m_1=pos(larr_1,iter)$，…，$m_p=(larr_p,iter)$.

这里，larri 是 C_i 的条件式的值形成的逻辑数组。令 $\mu=\min(m_1,m_2,\cdots,m_p)$，$C_i$ 是 C_1，…，C_p 序列中第一个使 pos 函数值为 μ 的 search－if 语句，则按 search－loop 定义的语义，(3) 可转换成：

```
        do   α  I = 1,iter
        G1
        ⋮
        Gi
Gi;     if(condi)  goto li
        Gi+1
        ⋮
        Gp+1
α       continue          (4)
```

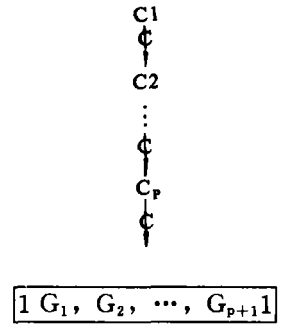


图4 循环(3)的相关

在(4) 中，记 G_1，…G_i 排成的赋值语句序列为 S_1，…，S_k；G_{i+1}，…，G_{p+1}为 S_{k+1}，…，S_n

则(4) 即为(2)。(证毕)

3 forward－loop 的循环重构

forward－loop 循环的本质特征是所包含的所有 if 语句均属于 forward－if 语句（无条件 goto 语句看作条件式恒为'true'的 if 语句)。现对 forward－loop 引入下述概念：

定义1 约束条件(guard) 在循环体内任何语句的执行都是在一定条件下约束执行的，该条件称为约束条件。约束条件可达到最“弱”，在循环迭代过程中其值恒为'ture'（被约束语句无条件执行)；或者达到最强（被约束语句为“死代码”)。

定义2 SGB (Same Guard Block) 具有相同 guard 的相邻赋值语句构成的语句块。从而语句的 guard 也是包含该语句的 SGB 的 guard.

本文目的是标识 forward－loop 的所有 SGB 并试图计算每个 SGB 的 guard，以便利用定理1转换循环中的控制相关为数据相关。

例 3

```
DO 100 I=1,100
IF(A(I).GT.O)  A(I)=0
A(I)=B(I)+T
IF(A(I).EQ.D(I))GOTO 50
Q(I)=A(I)-D(I)
D(I)=3
```

```
50          Q(I)=D(I)
100         CONTINUE
```

在循环体中标识SGB是十分简单而机械的过程。根据SGB的定义，并注意到循环内可能改变guard的因素，诸如if语句、无条件goto、语句号定义、else等等，易于标识出所有的SGB. 例3的四个SGB是：

$$\begin{aligned}&SGB1:\ A(I)=0\\ &\Omega\\ &SGB2:A(I)=B(I)+T\\ &SGB3:Q(I)=A(I)-D(I)\\ &\qquad D(I)=3\\ &SGB4:Q(I)=D(I)\end{aligned}$$

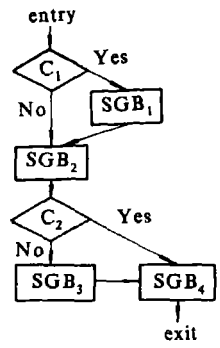


图5 例3的SGB控制流图

定义3 SGB控制流图F=〈N，A〉是描述SGB之间执行时序的有向图，其中，N为结点集，每个结点代表一个SGB或一个forward—if语句的条件式（分支结点）；A为弧集，对n_1，$n_2\in N$当且仅当n_1、n_2代表的实体之间有直接控制相关时，存在n_1指向n_2的弧（n_2控制相关于n_1）。直接控制相关系指两实体在执行时序中处理相邻位置。图5 给出例3的SGB控制流图。

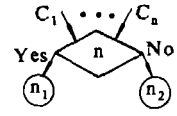


图6

在SGB控制流图上，对入度为零与出度为零的结点（二结点唯一）分别加上entry与exit标记；其次，在分支结点射出的两条弧上加了yes/no标志，表示条件式为‘真’/‘假’时的控制流向。这样的控制流图记录了求guard的所有信息。

由forward—loop的定义，控制流图不含回路。因此，从entry至exit必存在一个结点拓扑序。这就是本文求guard时需遵循的次序。

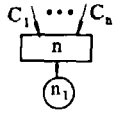


图7

分析控制流图后可发现，若结点n代表一个SGB，则n的出度必为1而其入度为任意正整数。一个SGB执行之后，其guard保持不变。故求出的guard可标记在SGB结点射出的唯一弧上。按循环的串行执行语义，从entry至exit的任意一条路径都代表着循环体控制流的一种可能的执行时序。这些时序依赖于条件式的真值。可归纳出求每条弧上时序条件的下述规则(时序条件即执行到某一点要满足的条件)：

(1) 与entry相联的弧其时序条件为‘true’；

(2) 对分支结点(图6)。令n的条件式为C，则弧(n,n_1)上的时序条件为$(C_1\vee C_2\vee\cdots\vee C_k)\wedge C$，而弧$(n,n_2)$的时序条件为$(C_1\vee C_2\vee\cdots\vee C_k)\wedge\bar{C}$；

(3) 对结点n，n_1(图7)，弧(n,n_1)的时序条件为：$(C_1\vee C_2\vee\cdots\vee C_k)$.

上述图中的圆型结点即可表示SGB，也可表示分支结点。由SGB射出的弧上的时序条件即该SGB的guard. 对图7的每条弧求时序条件得到图8.

上述方法求guard的过程中，随着分支结点的不断出现，guard逐渐变得复杂。文[6]给出了一般的布尔式化简算法，其复杂度与求解集合覆盖问题相当。

本文给出了一个简易的化简方法。

定义4 若控制流图的子图同时满足 a）、b），则称该子图为封闭子图：

a)子图中的结点存在拓扑序且拓扑序的首尾结点唯一确定，记为 n_i，n_j；

b)除 n_i，n_j 之外，子图之外的任何结点不与子图中的结点有弧相联，且与 n_i 相联的弧射入 n_i，与 n_j 相联的弧由 n_j 射出。

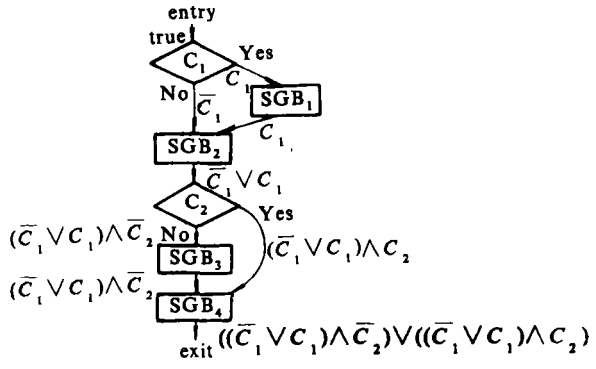


图8 求出 guard 的控制流图

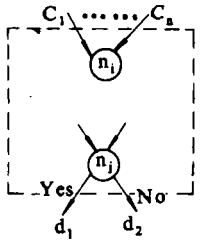


图9

封闭子图的重要性质是：射入 n_j（拓扑序首结点）的弧上的时序条件之逻辑和与射出 n_j（拓扑序尾结点）的弧上的时序条件之逻辑和等价。即：

当 $d_1=d_2$时（二弧重合，n_j 是 SGB），则 $d_1=d_2=(C_1 \vee C_2 \vee \cdots \vee C_k)$；

当 $d_1 \neq d_2$（即 n_j 是分支结点）时，$d_1=(C_1 \vee \cdots \vee C_k) \wedge C$，$d_2=(C_1 \vee \cdots \vee C_k) \wedge \overline{C}$ 这里，C 是 n_j 的条件式。

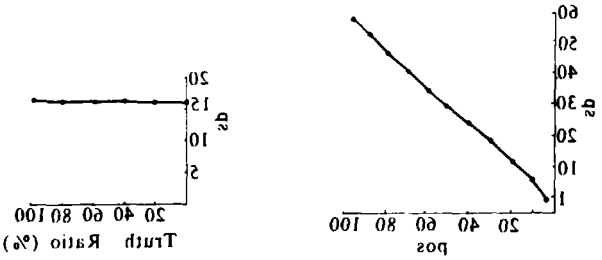
利用封闭子图的这一性质，可求得简化的 guard. 这一方法虽然不总是得到最简 guard，但由于求 guard 与化简可同时进行，因而本文的方法更有工程实用性。

将图8的时序条件化简，利用图中信息及定理1，将例7重构成：

```
      DO 200 I = 1,100
      C1 = A(I).GT.O
      A(I) = merge(0,A(I),C1)
      A(I) = B(I) + T
      C2 = A(I).EQ.D(I)
      Q(I) = merge(A(I) - D(I),Q(I),.NOT.C2)
      D(I) = merge(3,D(I),.NOT.C2)
      Q(I) = D(I)
200   CONTINUE
```

4 性能测试

if－loop 向量化的加速比与循环结构、条件式的真值密切相关。图10是例1、例2重构前后在 YH－1上测得的结果。从中可以看出，本文的方法具有工程实用价值。



(a)例1的性能曲线　　(b)例3的性能曲线

图10　对两个循环的性能测试

加速比(SP)＝标量执行时间/向量执行时间

真率(truth ratio)：逻辑数组中真值为'true'的百分比

参 考 文 献

1　吴少岩，吴健安. 非线性数组引用的向量化. 计算机工程与科学，1987

2　Chen H Bo, Ci Y Gui. Parallel Execution of Non－Do Loops. Proceedings of the 1987 International Colference on Parallel Processing, 1987: 512～516

3　Wolf M J. Optimizing Supercompilers for Supercomputers. Ph D Thesis University of Illinois at Urbana－Champaign, 1982

4　Polychronopoulos C D. Parallel Programing and Compilers. Kluwer Academic Publishers, 1988

5　Baner jee U. Dependence Analysis for Supercomputing. Kluwer Academic Publishers. 1988

6　McCLUSKEY E J. Jr Minimization of Boolean Functions. The Bell System Technical, 1956

Parallel Execution of Do－loops with Control Dependences

Wu Shaoyan　Wu Jianan

(Department of Computer Science)

Abstract

In this paper "merge" and "pos" functions which can be effectively implemented on modern vector machines are defined. By using the two functions the methods in which forward－if and search－if wthin do－loops are eliminated have been presented. The general method of vectorization for forward－loops is also discussed. Furthmore, an approach to simplify conditions (guards) of statements execution with the concept of Encapsulation Subgraph is proposed.

Key words　vector computer, do－loop, control dependence, guards, encapsulation subgraph