国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第11卷第4期　1989年12月　Vol.11 No.4

一种开发逻辑程序中AND并行性的静态编译方法*

黄志毅　胡守仁

（电子计算机系）

摘　要　逻辑程序AND并行性的开发是逻辑程序并行执行研究中的一个重要课题。文中提出了一种开发逻辑程序中AND并行性的静态编译方法。该方法分成三个阶段：第一阶段，进入模式（entry mode）的分析；第二阶段，退出模式（exit mode）的生成；第三阶段，执行图表达式（execution graph expression）的确定。通过运行一些基准程序可知，与其它方法相比，该方法能在“生产者—消费者”原则（producer—consumer scheme）下最大限度地开发AND并行性，且只需很小的动态开销。

关键词　逻辑程序设计，编译，并行执行，与（AND）并行性，数据相关性，PROLOG

分类号　TP31

在逻辑程序中，由于子目标间共享变量，使得与并行的开发变得相当困难。Pollard曾提出无条件地开发子目标间的与并行，然后对子目标所得到的解进行调解（reconcile）[5]。他的这种方法无疑能开发逻辑程序中的所有与并行，但它增加了系统的运算量，在实际应用中将得不偿失。目前，人们不再无条件地开发子目标间的与并行，而是基于“只有当子目标间不存在数据相关时才可并行执行”的“生产者—消费者”原则，提出各种各样的与并行开发方法[1,2,3,4]。在这些方法中，有一类方法要求程序员说明程序中的变量或子目标是否“生产者”或“消费者”，提供相应的一些信息[3,4]。我们认为，任何强迫或依赖程序员在程序级提供说明的与并行开发方法都不是普适的，它们改变了逻辑程序所具有的简单性、方便性等特点，不具有广泛性。在另一类方法中，有Conery(1983)在其AND/OR模型中提出的动态排序方法[1]。采用这种方法虽然能在“生产者—消费者”原则下开发最大限度的与并行，但动态开销很大。我们认为，通过对程序作静态分析来开发与并行的方法是一种降低动态开销的良好途径。Degroot(1984)首次提出了一种静动态结合的与并行开发方法[2]。通过对程序作静态分析和执行系统的动态支持，该方法能开发受限的与并行，并有效地降低动态开销。然而，Degroot仅限

*此项研究得到霍英东教育基金委员会资助

学术论文　1988年10月18日收稿

于在子句级对程序进行分析，无法清楚地了解程序的整体情况，因而导致所产生的执行图表达式中存在冗余的测试谓词[7]。此外，由于独立性算法的不完备性和mistyping，该方法中与并行的开发程度也受到较大的限制。我们认为，如果对程序进行深入的整体的分析，将能在静态编译时检查出子目标间的数据相关性，最大限度地识别与并行性，同时可以尽可能少地使用或不使用测试谓词，降低动态开销。本文从这一观点出发，提出一种较优的编译方法。该方法就开发与并行性而言，能达到Conery的方法所具有的效果，而且动态开销比Degroot的方法要小得多。

1 几个概念

1.1 输入和输出

在逻辑程序中，执行一个子目标时，若其中某变量X已被特例化，则称之为该子目标的输入变量；否则称之为输出变量。在一致化中，设有子句 $C: C^+ :- C^-$（C^+, C^- 分别为子句C的头部和体部）和询问：

$$? - A_1, \cdots, A_i, \cdots, A_k$$

此时选择子目标A_i与C^+进行一致化，得到MGU (most general unifier)θ. 若C^+中的任何变量X被θ特例化，则称该变量为子句C的输入变量；$C^+\theta$中的非特例化变量都称为子句C的输出变量。如果一个参数（项）中仅含输入变量，它就被称作输入参数(项)；否则，它就被称作输出参数（项）。

这里所说一个过程（子句）中的某个参数（变量）必须是输入的，即指在任何调用该过程（子句）的子目标与其过程头（子句头）进行一致化后，该过程头（子句头）中相应的参数（变量）必须是特例化的；否则，称该过程（子句）对相应的参数（变量）无输入要求。在逻辑程序中，若过程的某个参数或变量无输入要求，则它既可是输入的，也可是输出的。

1.2 进入模式(entry mode)和退出模式(exit mode)

一个子句的进入模式是指当一个子目标与它的子句头进行一致化后，该子句头各参数的特例化情况。一个子句的退出模式是指当该子句体中各子目标都求解完后，其子句头各参数的特例化情况。至于过程的进入/退出模式定义，可类似地得出（详见第2节）。

在模式中，用G和N分别表示相应参数是特例化的和非特例化的。显然，若子句某个参数必须是输入的，则它的进入模式必须为G；否则它的进入模式既可是G，也可是N。对于退出模式，可根据子句体中子目标执行完后各变量的特例化情况来确定。

1.3 与(AND)并行执行与数据相关性

在逻辑程序中，如果一个子句中两个以上的子目标并行执行，则称这些子目标的执行为与(AND)并行执行；如果两个子目标共享一个或多个未约束变量，则这两个子目标间存在数据相关。文中开发与并行所遵循的原则是：只有当子目标间不存在数据相关，它们才可并行执行。这也是通常所说的“生产者—消费者”原则。因此，准确地识别出子目标间的数据相关性，能够开发最大限度的与并行。

可将逻辑程序中的数据相关性分成两类：非特例化变量共享相关（uninstantiated

variable sharing dependency，简称UVSD）和非特例化变量耦合相关（uninstantiated variable coupling dependency，简称 UVCD）。在下文中，所说的子目标的变量通常是指在静态程序中出现在子目标中的变量。

定义 1　UVSD

两个子目标间存在UVSD，当且仅当这两个子目标共享一个以上未特例化变量。

定义 2　耦合效应（coupling effect）

1．若在一致化后，两个变量对应的约束项共享一个以上的非特例化变量，则称它们被耦合。例如，当p(x,y)与p(h(w),h(w))或p(z,f(z,w))一致化后，x和y被耦合。

2．若在一致化后，两个变量所对应的约束项分别含有一个相互耦合的变量，则称它们被耦合。例如，若z和w被耦合，则当p(x,y)与p(f(z),g(w))一致化后，x和y也被耦合。

定义 3　UVCD

两个子目标间存在UVCD，当且仅当这两个子目标分别有变量X，Y，且X和Y被耦合。

2　静态编译方法

在本节提出一种开发逻辑程序中 AND 并行性的静态编译方法。该方法分成三个阶段：进入模式的分析；退出模式的生成；执行图表达式的确定。该方法将一个逻辑程序看作一个有向图，每个过程是图中一个结点，每个结点有指针指向它所调用的过程的结点；若某个过程调用自身，则出现自圈。例如，可将例 1 中的程序转化成图 1 中的有向图。

例 1　快速分类程序

```
quicksort([__h^__t], __s):-
                split(__h, __t, __x, __y),
                quicksort(__x, __x1),
                quicksort(__y, __y1),
                append(__x1, [__h^__y1], __s).
quicksort([ ], [ ]).
split(__h, [__e^__t], [__e^__x], __y):-
                __e < __h,
                split(__h, __t, __x, __y).
split(__h, [__e^__t], __x, [__e^__y]):-
                __e >= __h,
                split(__h, __t, __x, __y).
split(__h, [ ], [ ], [ ]).
append([ ], __l, __l).
append([__h^__t], __l, [__h^__l1]):-
```

```
append(_t, _l, _l1).
? - quicksort([5,3,2,4,1,8,6,7,9,10], _1).
```

下面对上述三个阶段进行讨论。

2.1 进入模式的分析

该阶段旨在通过对程序进行分析，剔除一些不可能或不正确的进入模式。在逻辑程序中，若谓词的一个参数必须是输入的，则它决不能作为输出参数。因此，在对进入模式进行分析时，需要确定哪些参数必须是输入的。在本文中，采用一些启发式规则来确定每一子句和进程的进入模式。也可利用类似于模式生成的技术确定进入模式。但无论采用何种方法，必须保证正确性，即保证不将任何可能的进入模式剔除。

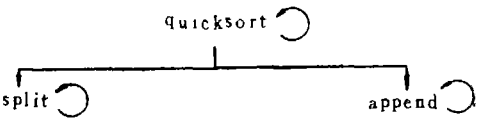

图 1 快速分类程序的调用关系图

下列规则是基于顺序PROLOG语义提出来的。不同PROLOG语义将导出不同的规则。在描述下列规则时，假设有变量X，子句头或体中的子目标S以及子句C，符号“X∈S”表示“变量X在S中出现”。规则如下：

(1) 内部谓词规则

一些内部谓词对其参数的I/O特性有特定的要求。如is谓词要求其右边表达式中变量必须是输入（特例化）；又如谓词“le”和“gr”都要求其参数中的变量是输入（特例化）。其它一些内部谓词也有类似的要求，可根据其具体的使用环境来确定其参数的I/O特性。

(2) 变量一致规则

若有变量$X\in C^-$，子目标$S_i\in C^-$，S_i非递归调用C^-，S_i中的X在C^-中首次出现，且S_i要求X必须是输入，则C^-中的某一个X也必须是输入。

(3) 递归谓词规则

若有子句：$p:-q_1,\cdots,q_{i-1},p,q_{i+1},\cdots,q_n$，子句头p中某一参数的一部分变量在$q_j(j=1,\cdots,i-1,i+1,\cdots,n)$中要求其为输入，而该参数的另一部分变量出现在子句体的p的相应位置，则该参数一定是输入。

(4) 若一个参数的所有变量必须是输入，则它必须是输入，且其进入模式必须是G.

(5) 非递归OR—bundle规则

若有一非递归过程的不同子句对参数有不同的进入模式要求，则将不同的进入模式进行“*”运算后得到该过程的进入模式，运算表如图2所示。从图2可以看出，对进入模式的要求是放宽的，这主要是从逻辑程序的多解特点考虑。“X”表示对进入模式无要求的情况。

*	G	X
G	G	X
X	X	X

图 2

•	G	X
G	G	G
X	G	X

图 3

(6) 递归OR—bundle规则

若一递归过程的不同子句对参数有不同的进入模式要求，则将不同的进入模式进行“•”运算后得到该过程的进入模式。运算表如图3。

从图 3 可以看出，此时对进入模式的要求是从严的。这是因为递归过程中不同子句将相互调用，必须保证每一子句的进入模式要求。

利用上述规则，从程序对应的有向图的终端结点开始，可由底至顶地分析每个过程。下面给出对例 1 中程序的分析结果：

quicksort的可能的进入模式：〈G,G〉，〈G,N〉

split的可能的进入模式：〈G,G,N,N〉，〈G,G,G,N〉，〈G,G,N,G〉，〈G,G,G,G〉

append在我们的规则下对进入模式无要求。

须指出的是：上述规则不一定是完备的，即它不一定能剔除所有不可能的进入模式，同时也没有考虑隐含递归的情况；不过它们是正确的。当然，也可对程序作进一步的分析，找到更完备的规则。但规则的完备性只影响下一阶段的工作量。本编译方法独立于任何具体的规则。

2.2 退出模式的产生

该阶段旨在对每个过程的每种可能的进入模式产生其相应的退出模式。仍将逻辑程序看成一个有向图，自底向上地对它进行分析。由于是自底向上分析，所分析的过程的每一子句，其体中的子目标的进入/退出模式的分析已经完成（递归情况除外）。在对过程分析时，需对过程的每一子句的不同进入模式进行下述算法的分析。

退出模式生成算法：

输入：一个子句C：$C^+ - C^-$及其进入模式

输出：相应的退出模式

变量：GV，当前特例化变量集；i，子句体中子目标序号；S，当前子目标；NG，已经执行的子目标中的非特例化变量集；SG,SN：分别对应子目标S执行后S的特例化变量集和非特例化变量集。

(1) 初始化

i：=1，NG：=[ ]，

GV：={进入模式为G的参数中的变量}

(2) 若C^-=[ ]，则C为事实，转6。

(3) 置S为C^-的第i个子目标。

(4) 若S=nil，转6。

(5) 对S执行如下步骤：

i) 根据GV确定S的进入模式。

ii) 根据2.1节中的进入模式分析结果判S的进入模式是否符合要求。若不符合，说明C^+的进入模式不正确。打印出错信息，结束。

iii) 若S递归调用C^+，则根据S的进入模式和该过程中事实的进入/退出模式分析，确定S的退出模式，否则，根据S的进入模式和过程S的进入/退出模式分析确定其退出模式。

iV) 根据退出模式确定S中的特例化变量集SG和非特例化变量集SN.

v) GV：=GV∪SG

vi) 若NG∩SG≠nil，

则，i:=0，NG:=[ ]；

否则，NG:=NG∪SN

vii) i:=i+1，转3．

(6) 根据GV确定子句C的退出模式，结束。

U	G	N
G	G	N
N	N	N

图 4

对于每个子句的每一进入模式，可产生其退出模式。而就一个过程而言，同一进入模式对不同子句可能产生不同的退出模式。对于这些不同的退出模式，进行了“∪”运算。运算表如图4．

从图4可以看出，所采用的原则是安全的，即若过程的某个参数由进入模式N变成退出模式G，则该过程一定是该参数中变量的真正生产者。

对于例1中的程序，可利用上述算法生成如下结果（表1～表3）。

表1 Entry/exit mode table for “quicksort”

entry mode	exit mode
<G, G>	<G, G>
<G, N>	<G, G>

表2 Entry/exit mode table for “append”

entry mode	exit mode
<N, N, N>	<N, N, N>
<N, N, G>	<G, G, G>
<N, G, N>	<N, G, N>
<N, G, G>	<G, G, G>
<G, N, N>	<G, N, N>
<G, N, G>	<G, G, G>
<G, G, N>	<G, G, G>
<G, G, G>	<G, G, G>

表3 Entry/exit mode table for “split”

entry mode	exit mode
<G, G, N, N>	<G, G, G, G>
<G, G, G, N>	<G, G, G, G>
<G, G, N, G>	<G, G, G, G>
<G, G, G, G>	<G, G, G, G>

2.3 执行图表达式的确定

本阶段旨在对子句的每一进入模式产生其子目标的执行顺序或执行图表达式。在进行以上的进入/退出模式分析后，可以分析出子目标间的数据相关性。对于第一类数据相关(UVSD)，可根据变量的特例化情况来予以确定；对于第二类数据相关(UVCD)，只有当两个变量都未特例化时才可能存在，无法在静态时确定。此时需要采用Degroot提出的动态测试谓词IPAR，以确定具有不同特例化变量的子目标间是否可以并行执行。这样就可确定出子目标间的执行顺序及并行关系。

如果要提出一个具体的执行图表达式产生算法，必须首先确定应保持何种PROLOG语义、是否考虑付作用 (side—effect)问题[6]。本文所提出的编译方法独立于具体的执行图表达式生成算法。必须说明的是，执行图表达式中子目标必须附带一个进入模式，以使程序执行时能根据其进入模式找到被调用子句的正确的执行图表达式。执行图表达式中所用到的PAR，SEQ、IPAR等符号的意义与Degroot所提出的相同。下面给出基

于[11]中算法的例1中程序的执行图表达式

```
(quicksort([_h^t],_s),GG):-
        (SEQ(split (_h,_t,_x,_y),GGNN)
            (SEQ (PAR (quicksort(_x,_x1),GN)
                      (quidksort(_y,_y1),GN))
            (append(_x1,[_h^_y1],_s),GGG))).
(quicksort([_h^_t],_s),GN):-
        (SEQ(split(_h,_t,_x,_y),GGNN)
            (SEQ (PAR (quicksort(_x,_x1),GN)
                      (quicksort(_y,_y1),GN))
            (append(_x1,[_h^_y1],_s),GGN))).
(split(_h,[_e^_t],[_e^_x],_y),GGNN):-
        (PAR(_e < _h,GG)
            (split(_h,_t,_x,_y),GGNN)).
(split(_h,[_e^_t],[_e^_x],_y),GGGN):-
        (PAR(_e < _h,GG)
            (split(_h,_t,_x,_y),GGGN)).
(split(_h,[_e^_t],[_e^_x],_y),GGNG):-
        (PAR(_e < _h,GG)
            (split(_h,_t,_x,_y),GGNG)).
(split(_h,[_e^_t],[_e^_x],_y),GGGG):-
        (PAR(_e < _h,GG)
            (split(_h,_t,_x,_y),GGGG)).
(split(_h,[_e^_t],_x,[_e^_y]),GGNN):-
        (PAR(_e >= _h,GG)
            (split(_h,_t,_x,_y),GGNN)).
(split(_h,[_e^_t],_x,[_e^_y]),GGGN):-
        (PAR(_e >= _h,GG)
            (split(_h,_t,_x,_y),GGGN)).
(split(_h,[_e^_t],_x,[_e^_y]),GGNG):-
        (PAR(_e >= _h,GG)
            (split(_h,_t,_x,_y),GGNG)).
(split(_h,[_e^_t],_x,[_e^_y]),GGGG):-
        (PAR(_e >=_h,GG)
            (split(_h,_t,_x,_y),GGGG)).
```

3　实验结果与性能分析

基于上述方法，可以产生有效的执行图表达式，以开发逻辑程序的与并行。为了验

证该方法的有效性，在SES—PIM模拟实验系统[8]中实现了基于该方法的预编译器。为了进行对比实验，还实现了基于Degroot方法的预编译器。在PSOF模型[9]下分别采用了这两种方法，并与在PSOF模型中曾经采用的动态方法[10]进行了对比。通过运行一些典型程序，如矩阵乘，快速分类，迷宫问题，皇后问题，汉诺塔，集合运算等，得到了一些有意义的结果。下面运行结果列表给出。

表 4 在SES—PIM系统中执行各程序的测试谓词数

		Degroot方法	本文的方法
快速分类	IPAR	38	0
	GPAR	38	0
n后问题	IPAR	72	0
	GPAR	176	0
矩阵乘	IPAR	12	12
	GPAR	12	0
汉诺塔	IPAR	0	0
	GPAR	31	0
集合运算	IPAR	190	0
	GPAR	190	0
迷宫问题	IPAR	0	0
	GPAR	30	0
求阶乘	IPAR	0	0
	GPAR	19	0
总计	IPAR	312	12
	GPAR	496	0

表 5 在SES—PIM系统中执行各程序时的并行度

	Degroot方法	本文的方法	动态方法
矩阵乘	6.08	6.08	6.08
快速分类	1.83	3.82	3.82
求阶乘	1.62	6.14	6.14
汉诺塔	1.90	5.21	5.21
n后问题	19.21	19.21	19.21
迷宫问题	10.59	10.59	10.59
集合运算	11.04	61.15	61.15
平均	7.1	16	16

根据实验结果，可得如下结论：

(1) 本文提出的方法在大多数情况下能最大限度开发AND并行。根据所运行的基准程序统计，该方法开发的平均并行度为16，与动态方法相同，且是Degroot方法所开发的并行度的2倍(7.4)。

(2) 该方法具有极少的动态开销。通过运行上述基准程序，作者统计出：该方法使

用的测试谓词为12次IPAR，0次GPAR；而Degroot方法使用了312次IPAR，496次GPAR，是该方法的67倍。

（3）避免了冗余测试谓词。文中区分了变量的特例化和非特例化两种情况；在使用IPAR测试谓词时，变量一定是非特例化的，且必须在运行时检测，因此IPAR测试是必需的。

本方法不足之处在于编译时间较长，是Degroot方法的4倍。今后对预编译器优化时，将在并行度和编译开销之间进行权衡，以达到整体的性能价格比最优的目的。

4 结束语

文中提出了一种开发逻辑程序与并行的静态编译方法。该方法通过在程序级作静态数据相关性分析，能够最大限度地开发出AND并行性，且只需极小的动态开销。该方法涉及到的一些技术需要作者今后作进一步的研究。

致　　谢

本文成文过程中，孙成政博士提出很多有益的意见和建议，本课题组成员高耀清、王平同志与作者进行了极富启发的讨论，在此一并表示感谢。

参 考 文 献

[1] Conery J S. The AND/OR model for parallel interpretation of logic programs. Ph.D.Th., Dept.of Infor. and Computer Sci., Univ.of California, Irvine, 1983

[2] DeGroot D. Restricted. And—Parallelism. Proc. of the Inter. Conf. on Fifth Generation Computer System, 1984;5:471～478

[3] Clark K L. and Gregory S. PARLOG:Parallel Programming in Logic. Research report DOC 84/4, Imperial College, London, England

[4] Shapiro E Y. A Subset of Concurrent Prolog and its Interpreter. ICOT, Technical Report:TR—003 (Feb., 1983)

[5] Pollard G H. Parallel Execution of Horn Clause Programs. Ph.D.Th., Univ. of London, Imperial College of Sci. & Tech., U.K., 1987

[6] DeGroot D.Restricted And—Parallelism and Side—effects in Logic Programming. in Supercomputers and AI Machines, Kai Hwang and Doug DeGroot Editors, McGraw—Hill, 1988

[7] 黄志毅，胡守仁. 逻辑程序数据相关性分析——RAP模型分析. 第二届软件行业协会人工智能会议, 1988

[8] Sun Chengzheng and Ci.Yungui. SES—PIM:A Simulation and Experiment System for PIM—PSOF. the Second National Conference On Logic Programming, China 1986

[9] Sun Chengzheng and Ci Yungui. PSOF:A Process Model Based on the OR—forest Description. Proc. of the Inter. Conf. on Computer and Communication, Beijing, 1986

[10] Sun Chengzheng and Ci Yungui. An Automatic partition Algorithm for AND—

Parallel Execution in the Framework of OR—forest. Proc. of the Second Inter. Conf. on Computers and Applications, Beijing, 1987:223~229

[11] Hwang Zhiyi and Hu Shouren. A Compiling Approach for Exploiting And—Parallelism in Parallel Logic Programming Systems. Proc. of Parallel Architecture and Language Europe, Netherlands, 1989

A Compiling Approach for Exploiting AND-parallelism in Logic Programs

Huang Zhiyi Hu Shouren

(Department of Computer)

Abstract

Exploiting AND-parallelism is important in the research of parallel execution of logic programs. In this paper, a compiling approach for exploiting AND-parallelism in logic programming is presented. The approach consists of three phases: analysis of entry modes; derivation of exit modes; and determination of execution graph expressions. Compared with other approaches, this approach, with the compile-time program-level data dependence analysis of logic programs, can efficiently exploit AND-parallelism in logic programs. Two precompilers, based on our approach and DeGroot's respectively have been implemented in SES-PIM system. Through compiling and running some typical benchmarks in SES-PIM, we conclude that our approach can exploit the maximum AND-parallelism under "producer-consumer" scheme, exactly the same degree as the dynamic approach once employed in SES-PIM, and needs significantly less dynamic overhead than DeGroot's while exploiting more AND-parellelism than DeGroot's.

Key words: Logic programming; compiling, parallel execution, andparallelism, data-dependence, PROLOG