PARLOG 顺序编译实现技术

郑纬民 杨和平

(清华大学)

A SEQUENTIAL COMPILING TECHNIQUE FOR PARLOG

Zheng Weimin and Yang Heping

(*Tsinghua University*)

ABSTRACT

PARLOG relations are divided into two types: single-solution relations and all-solution relations. A sequential compiling technique for PARLOG single-solution relations, which based on and/or tree model and process-scheduling strategy, is introduced in this paper.

摘 要

PARLOG 语言是一种并行逻辑程序设计语言。其关系定义分为单解关系和所有解关系。本文着重讨论PARLOG 单解关系基于与/或树模型和进程调度方法的顺序编译实现技术。

§1. PARLOG 顺序编译系统的体系结构

PARLOG 是一种适合于并行计算的逻辑程序设计语言[1、2], 它同时采用了“与并行”和“或并行”计算模型，并引入了多种并行性控制机制。PARLOG 语言中的模式(mode) 说明定义了程序中的数据相关性，因而复杂的合一操作可用简单的输入/ 输出匹配实现。PARLOG 采用警卫Horn 子句(Guarded Horn Clauses) 和确认选择非确定性(Committed Choice Nondeterministic) 语义，大大限制了程序中的或并行计算，使得子句的选择无回溯。另外PARLOG 还同时采用了“并行与”、“串行与”、“并行或”、“串行

1989 年10 月9 日收到，1990 年3 月26 日定稿。

或”操作语义，既限制了计算的并行性，又提供了程序设计的灵活性。PARLOG 的这些特点使得PARLOG 易于获得高效实现。

本文介绍一个PARLOG 顺序编译系统。该系统以C 作为目标语言，采用进程调度方法，实现了PARLOG 单解关系的编译。而PARLOG 的所有解关系则可通过PARLOG 构造子set 和subset 实现。编译系统的体系结构如图1 所示。系统可分为三部分：①PARLOG 到KP 及KPaot 的编译；②KPaot 到目标代码的编译；③执行系统和内部调词库。在执行系统中，除包含合一基元、I/O 操作及其它实现级基元外，还包括进程调度算法、对挂起进程的处理算法、存储管理和垃圾收集算法。该系统已运行于S 1280 和SUN 3/260 机器上。

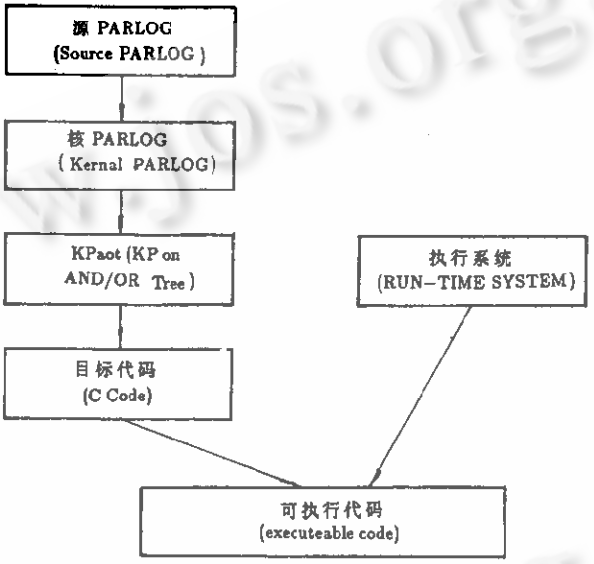

图1　编译系统的结构

§2. KP、KPaot 的特点及PARLOG 到KP 和KPaot 的编译

KP 和KPaot[1、2] 是两种同PARLOG 具有相同表达能力且形式更为简单的中间语言，它们可以看成是两个不同层次的PARLOG 抽象机。

2.1 KP 的特点及PARLOG 到KP 的编译

KP 是PARLOG 的一个具有同等描述能力的子集。在KP 中，没有显式的模式说明，子句头部关系的所有参量均为互不相同的变量，并且所有的输入/ 输出合一都由简单的输入匹配“<=”、输出匹配“: =”、相等匹配“==”基元显式表示。

PARLOG 程序编译成KP 代码时，各个子句的编译是相互独立的。编译一个子句时，首先将子句头部关系的各参量均换名为互不相同的变量，然后依据子句的模式说明，分别用输入匹配、输出匹配、相等匹配操作来简化子句中的合一操作，并把输入匹配和相等匹配作为子句的卫士条件，把输出匹配作为子句的体条件，从而得到KP 形式的子句。

例如，设有过程merge, 其模式说明为merge(list1?, list2?, list^), 那么子句

merge ([u|x], y, [u|z])<—merge (x, y, z)

经编译生成如下形式的KP 子句:

merge (p1, y, p3) <—[u|x]<=p1: p3:=[u|z]&merge (x, y, z)

在PARLOG 到KP 的编译过程中，程序的提问式不作任何变化。

2.2 KPaot 的特点及KP 到KPaot 的编译

KPaot 是KP 基于与/ 或树模型的表示形式，它更易于机器实现。在KPaot 中，子句的卫士条件是由卫士指令和卫士关系调用组成的，其形式为:

<Guard_ins1&...&Guard_insj>&<Guard_cond1, ..., Guard_condk>

而子句的体条件是由体指令和体关系调用组成的，其形式为:

<Body_ins1&...&Body_insm>&<Body_call1, ..., Body_calln>

其中，卫士指令和体指令都是十分简单的操作，它们可以立即执行而无需创建进程。而且子句中卫士关系调用和体关系调用均为并行合取式。

从KP 编译到KPaot 代码时，各子句的编译也是独立的。编译一个子句时，首先检查KP 中卫士的安全性，把不安全卫士中对环境的赋值操作编译成子句体部的输出匹配，并进一步编译成体指令，因而卫士条件的执行不会产生对环境的约束。第二步，将程序中的串行合取式编译成并行合取式，并通过设置同步变量来实现其串行操作语义。例如，串行合取式(p&b) 可编译成并行合取式(call(p, s), next(s, b)), 其中s 是同步变量。仅当p 执行成功时，s 的值方为真，此时关系b 才能执行。否则next(s, b) 失败或挂起。第三步，对子句的卫士条件和体条件分别进行粒度化简，尽可能地把各种操作编译成卫士指令和体指令，得到的KPaot 代码只含有卫士指令、体指令和用户定义的关系调用(包括内部谓词)。上节中的例句merge 经进一步编译生成如下形式的KPaot 子句:

merge (p1, y, p3)<-data(p1)&get_list(p1, u1, x)&u: =u1:
put_variable(z)&put_list (l,u, z)&p3: =l&merge(x, y, z)

对KP 中的提问式进行编译时，只需按上述第二步和第三步所示过程进行编译，便可得到相应的KPaot 提问式。

§3. KPaot 到目标代码的编译

从KPaot 抽象机代码编译成目标代码时，关键问题是如何实现PARLOG 的确认选择非确定语义和或并行性控制机制。本文采用随机数技术实现KPaot 的确认选择非确定性语义。

3.1 对KPaot 子句的编译

编译系统中，每一个KPaot 关系均被表示为一个进程结构结点，而KPaot 的参量则用参量结点表示。参量结点具有PARLOG 的参量特性，而不同于传统语言中的数据类型。每一个KPaot 子句都可以编译成一个目标语言的函数(称为功能函数), 而每一个功能函数的执行即为一个活动进程。在功能函数中，子句的卫士条件总是最先得到执行。如果卫士条件的执行失败，那么该功能函数立即失败返回，表明该子句为非候选子句；若卫士条件的执行挂起，则该功能函数的执行挂起，表明该子句为挂起子句；否则，若卫士条件执行成功，那么子句的体指令将立即获得执行。仅当子句的体指令执行成功时，功能函数才创建子进程来计算体关系调用合取式，否则功能函数失败返回，表明相应进程(目标) 的执行失败。如果一个功能函数的卫士条件、体指令和子进程都执行成功，那么该

功能函数成功返回，表明对相应目标的计算获得成功。

3.2 对KPaot 过程的编译

由于KPaot 子句均被编译为功能函数，因此根据PARLOG 的确认选择非确定性语义和过程中固有的或并行性控制操作，KPaot 中的任一过程都可以编译为对这些功能函数的调用，这种调用实现了KPaot 的候选子句搜索算法。这里采用了随机数技术来实现对并行子句组的搜索算法，并且避免了对同一个子句的重复搜索。

3.3 对KPaot 提问式的编译

KPaot 提问式由体指令和用户定义的关系调用组成，并且用户定义的关系调用都是并行合取项。因此编译KPaot 提问式时，提问式中的体指令被编译成为可以立即执行的指令序列，而提问式中用户定义的关系调用则被编译成初始化进程池，每一个关系调用对应于池中的一个进程。所有进程都是并发的，因此进程池是平坦的。程序的执行从这些进程的执行开始。

上述算法将另撰文讨论。

§4. PARLOG 执行机制的实现

在传统机器上实现PARLOG 语言的编译，其首要问题是如何实现PARLOG 的并行计算语义。上节已讨论了PARLOG “或并行性”计算语义的实现，这节讨论PARLOG “与并行性”计算语义的实现及挂起进程的处理算法。

本文采用进程调度方法来实现PARLOG 的“与并行”计算语义。为了获得高效实现，采用深度优先算法调度进程，采用busy-waiting 策略处理挂起。深度优先算法同busy-waiting 策略相结合，程序将尽可能完全地执行，产生大量可用数据，使挂起进程的数目减少到最低限度，因而这种策略具有高效性。但是这种纯粹的深度优先策略有时会导致不停机计算，为解决这一矛盾，可将深度优先策略和宽度优先策略结合起来，对程序的执行深度加以限制，当程序的执行深度超过一定限制时，按宽度优先策略调度进程。这种策略在一定程度上解决了不停机运算问题。

另外，为了高效实现PARLOG 编译，还采用了动态存储管理和垃圾收集技术，并扩充了PARLOG 操作原语，这些将另撰文讨论。

参考文献

[1] Clark K.L and Gregory S., “PARLOG: a Parallel Logic Programming Language” acm Transaction on Programming Language and System, Vol. 8, No. 1, Jan. 1986, pp. 1-46.

[2] Gregory S., “Design, Application and Implementation of a Parallel Logic Programming Language”, Ph.D thesis, Dept. of Computing, Imperial College, LONDON, Sep. 1985.

[3] Steve Gregory, “Parallel Logic Programming in PARLOG”, Addison-wesley Publishing Company, 1987.