KD—PARPRO：一个基于知识的并行化工具——总体设计与功能描述

金国华 陈福接*

（长沙工学院计算机系，长沙 410073）

KD—PARPRO: A KNOWLEDGE—BASED PARALLELIZING TOOL

Jin Guohua and Chen Fujie

(*Department of Computer Science, Changsha Institute of Technology, Changsha* 410073)

Abstract The advances in parallel processor architectures require the corresponding supports from software tools. Recently, parallelizing tools are extensively and deeply researched in the world. However, only a small fraction of the available parallelism in programs can be exploited with existing techniques because of the difficulties of interprocedural dependence analysis, the overdependence optimizing criterion on machine features, the overheads of synchronization, communication and scheduling, etc. Futhermore, as a tool, it should include a user—friendly interface and have good transportability and expendability. In this paper, we describe the design of KD—PARPRO, a knowledge—based parallelizing tool and functions of the heuristic translator. Finally, we introduce a dynamic control technology.

摘要 并行处理系统结构的发展要求相应的软件工具的支持. 目前，国际上对并行化工具正开展广泛深入的研究，但由于过程间相关性分析难，最优化判别标准过分依赖于机器特性，同步通讯问题及调度开销大等原因，使得现有技术所能开发的并行性极为有限. 另外，对软件工具的要求不但是正确性，还要有友好的用户界面和良好的可移植性，可扩充性. 本文讨论了基于知识的并行化工具 KD—PARPRO 的设计思想，对启发式转换器进行了功能描述，并介绍了动态控制技术.

* 本文 1990 年 12 月 27 日收到，1991 年 6 月 5 日定稿. 本文是国家自然科学基金资助课题. 作者**金国华**，博士生，助教，主要研究领域为并行化编译，同步通讯机制，相关性分析理论. **陈福接**，教授，博士生导师，主要研究领域为并行处理系统结构，并行编译，多机操作系统，并行算法，外围子系统.

§0. 引　言

并行处理系统结构已得到了很大发展，这为超高速运算提供了可能性，然而，要实现这个可能性，要提高实际运算速度，充分发挥机器的潜在性能，还需要相应软件工具的支持，其中包括在保证合理的运算开销下，极大限度地开发程序的并行性. 目前，向量化技术已基本成熟[1][2][6]，并得到了广泛关注[3][5]，但现有技术所能开发的并行性极为有限. 实践证明，并行性的开发是一个非常复杂而又艰苦的工作，原因可归结为如下几点：

(1)相关性分析，尤其是跨过程相关性分析十分困难，为保证程序的正确性，编译对一些未知程序特性只能作保守估计.

(2)最优判断标准是一个非常关键但很难确定的问题.

(3)实现并行化所带来的同步通讯、调度开销和存储器、网络冲突所引起的额外随机延迟，严重影响了性能的发挥.

另外，我们感到，作为一个好的并行化软件工具应具备如下特性：

(1)良好的用户界面，利用交互功能，用户可以询问并行化进展，参与整个并行化过程.

(2)可移植性，工具应该具有通用性，应适用于各种机器结构.

(3)可扩充性，可按用户要求对工具作功能扩充.

针对以上分析，我们提出了一种基于知识的并行化处理工具 KD－PARPRO，本文重点讨论其设计思想和主要功能，并介绍动态控制技术.

§1. 设计思想

1. KD－PARPRO 的总体结构

KD－PARPRO 利用启发式转换和动态控制来最大限度地开发程序并行性，总体结构如图 1 所示.

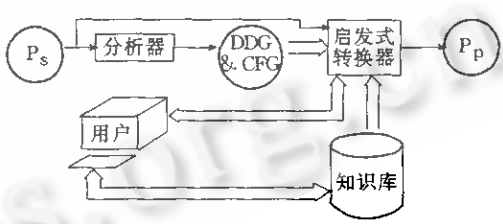

图1　KD－PARPRO 总体结构

(1)分析器对输入的串行代码进行分析，产生数据相关图 DDG 和控制流图 CFG；

(2)启发式程序转换器根据分析得到的程序特性，运用知识库提供的机器知识和程序转换知识选择一个最优的转换序列$\langle t_1 t_2 \cdots t_n \rangle$将串行代码 Ps 转换成具有动态控制结构的并行优化代码 Pp；

(3)机器特性、事实、启发式信息等以规则形式放入知识库中，启发式信息采用多层结构.

(4)用户界面采用菜单选择和图形显示功能实现用户和分析器，启发式转换器及知识库的交互功能.

由于引入了知识库概念，使得并行化对具体机器的依赖仅表现在知识库规则上，这样，在不同机器上使用工具 KD－PARPRO 时只需修改知识库中的规则，而扩充并行化处理能力也可表现在增加库中的规则上，使得工具具有极好的可移植性和可扩充性.

2. 并行性开发过程的研究

并行性的开发过程可以划分为如图 2 所示的三个阶段.

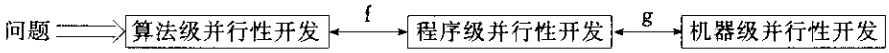

图2 并行性开发过程

整个过程涉及到两个级间双向映射 f,g. 映射 f 所解决的是:语言应提供什么样的并行程序结构才能完整地表达算法的并行性和算法应如何调整来充分利用现有语言所提供的并行结构. 映射 g 所完成的任务是:机器应提供什么样的并行体系结构来全面支持各个级别的程序并行性的开发和机器结构如何修改程序结构以充分利用机器所提供的潜在并行性.

我们这里关心的是如何最大限度地开发程序并行性,找到程序结构和机器结构的最优匹配,以确保程序并行性的充分实现.

程序的结构是通过语言定义的,它决定了程序所具有的并行性,同样,我们可以用计算模型来表示机器的结构,代表机器所具备的并行性,定义

$$\Phi_c=\Phi_{f_1}\times\Phi_{f_2}\times\cdots\times\Phi_{f_n}=\{\varphi_c\mid\varphi_c=(\varphi_{f_1},\varphi_{f_2},\cdots,\varphi_{f_n}),\varphi_{f_i}\in\Phi_{f_i},i=1,2,\cdots,n\}$$

为所有计算模型的集合,φ_c 为某特定机器的计算模型,Φ_{f_i} 表示机器特征空间,φ_{f_i} 为 f_i 特性取 Φ_{f_i} 中某特定值,这里的机器特性包括:结构特性,处理机特性,存储器特性和网络特性等. 结构特性又分为处理机数目,计算模式(SIMD,MISD,MIMD)等;处理机特性包括 CPU 速度,向量寄存器数目和长度,向量指令类型,向量起步时间,向量链接特性,scatter—gather 特性等;存储器特性含 cache,LM 的容量和共享模式,MM 特性等;网络特性有网络拓扑,网络带宽,级延迟,路由策略等.

我们的目标是要找到一组最优的变换序列 $t_1t_2\cdots t_n$,使得下列变换成立:

$$t_1t_2\cdots t_n(\varphi_c,P_s)\rightarrow P_p,t_i\in T\ (i=1,2,\cdots,n)$$

T:程序变换集合;φ_c:某种机器结构的计算模型;P_s:串行代码;P_p:含有动态控制结构的并行代码.

这里涉及到两个非常难以解决的问题:

(1)何时应用何变换最优. 因为不同的变换序列将得到不同的结果,同样一个变换在不同的程序状态下应用效果不同;

(2)系统如何判别某程序区域对特定的机器结构是最优的并停止变换过程.

由于测试是否可以进行某个变换和变换本身都需花费很大代价,且对于一个给定程序区域,可以有很多不同的变换序列,要通过试验所有变换序列然后选择最优序列来优化程序是不现实的. 和人工智能最优解的求解需要启发式搜索一样,在这里引进启发式信息来保证每一步变换的最优性同样是有益的. 我们引入所谓加权匹配函数 M 来描述程序结构和机器结构的匹配程度,即最优程度:

$$M(\varphi_c,P)=\lambda_1M_1(\varphi_c,P)+\lambda_2M_2(\varphi_c,P)+\cdots+\lambda_nM_n(\varphi_c,P)$$

$$\sum_{i=1}^{n}\lambda_i=1\qquad 0\leqslant\lambda_i\leqslant1,\qquad 0\leqslant M_i\leqslant1,\qquad i=1,2,\cdots,n$$

$M_i(i=1,2,\cdots,n)$ 为各匹配因素,$\lambda_i(i=1,2,\cdots,n)$ 为相应匹配因素的优先级.

用户界面为用户提供交互手段控制最终匹配程度,$(0\leqslant\epsilon\leqslant1)$(当 $M(\varphi_c,P)\geqslant\epsilon$ 时终止

变换过程)并决定各匹配因素的优先级大小,如:$\lambda_i<\lambda_j$,表示 M_j 因素比 M_i 因素更重要;$\lambda_i=0$,表示可不考虑 M_i 因素;$\lambda_i=1,\lambda_j(j\neq i)=0$,表示只考虑 M_i 因素.

§2. 启发式转换器的功能描述

在选择最优转换序列时我们利用了启发式信息,启发式转换器的功能描述如图3所示.

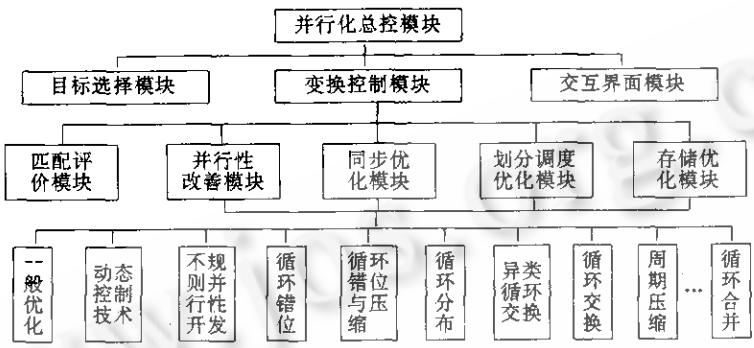

图3 启发式转换器的功能描述

(1)并行化总控模块控制整个程度的优化过程.它可以反复调用目标选择模块,对特定程序区域或整个程序进行优化.

(2)目标选择模块选择需优化的程序区域,根据程序的大小、结构和优化程度的不同,程序区域可小到循环,大到整个程序.

(3)变换控制模块通过各种转换程序来实现四种类型的优化,并调用匹配评价模块决定以何种优先次序对程序区域或程序执行这四种优化.四种优化分别在并行性改善模块,同步优化模块,划分调度优化模块,存储优化模块中完成.并行性改善模块用于改善程序结构,减少或消除数据或控制相关性;同步优化模块负责减少处理机间的同步通讯开销;划分调度优化模块用于对程序进行划分并进行处理机调度以取得各处理机间的负载平衡;存储优化模块的目的在于利用 cache、LM,向量寄存器等快速存储部件,消除不必要的数据访问,缩短访问时间,加快计算速度.启发式转换器是 KD-PARPRO 的核心,其工作量很大.

§3. 动态控制技术

利用动态控制技术目的有两个:(1)动态控制技术对运行时实际可能发生的各种情况产生出多种相应最优的程序代码供运算时选择;(2)通过动态控制技术,利用运行过程中所获得的信息,可以识别出更多的不相关成份,开发更多的并行性.下面用两个例子来分别说明动态控制技术的这两个重要用途.

例1:
```
DO  I=1,N
    X=A(I)+B(I)
    C(I)=X*X
ENDDO
```

利用标量膨胀技术可以消除迭代间的数据相关性,产生向量、DOALL、DOALL—向量三种形式的并行代码:

向量代码:

```
X(1:N)=A(1:N)+B(1:N)
```

```
C(1:N)=X(1:N)*X(1:N)
```

DOALL 代码:

```
DOALL I=1,N
   X(I)=A(I)+B(I)
   C(I)=X(I)*X(I)
ENDDOALL
```

DOALL—向量代码:

```
DOALL I=1,P
   X(I:N:P)=A(I:N:P)+B(I:N:P)
   C(I:N:P)=X(I:N:P)*X(I:N:P)
ENDDOALL
```

四种代码结构(包括标量代码)都存在各自的开销,设 β 为循环体执行时间,l 为循环内指令数,每一指令 S_i 只有一个操作 OP_i($i=1,\cdots,l$),OP_i 的流水线级数为 n_i,CP 为 CPU 时钟周期,P 为空闲处理机数,t_s 为向量指行取指、取数开销,VL 为向量长度,σ_0 为 DOALL 执行开销系数,又设 T_s,T_v,T_p,T_{pv}分别为标量,向量,DOALL,DOALL—向量代码执行时间,并且不考虑向量链接功能,则:

$$T_s=\beta\cdot N$$

$$T_v=l\cdot t_s+\frac{N}{VL}\left[CP\cdot\sum_{i=1}^{l}n_i+l\cdot(VL-1)\cdot CP\right]$$

$$T_p=\frac{N}{P}\cdot\beta+\sigma_0 P$$

$$T_{pv}=\frac{l\cdot t_s}{P}+\frac{CP\cdot N}{VL\cdot P}\left[\sum_{i=1}^{l}n_i+l\cdot(VL-1)\right]+\sigma_0 P$$

通常开销时间以标量→向量→DOALL→DOALL—向量呈增长趋势,但其并行程度也同样呈增长趋势,当 N,P 足够大时,并行度的增长所带来的好处将足以抵消开销所带来的缺陷,通过解:

$T_v<T_s$

$T_p<T_v$,$T_p<T_s$

$T_{pv}<T_p$,$T_{pv}<T_v$,$T_{pv}<T_s$

可分别得到 N_1,N_2,N_3使:

$N\leqslant N_1$时,标量代码最优

$N_1<N\leqslant N_2$时,向量代码最优

$N_2<N\leqslant N_3$时,DOALL 代码最优

$N>N_3$时,DOALL—向量代码最优

在确定了 N_1,N_2,N_3的值以后可以产生如下带有动态控制的代码:

```
IF (N.LEQ.N1) THEN
   DO I=1,N
      X=A(I)+B(I)
      C(I)=X*X
   ENDDO
ELSE IF (N.LEQ.N2) THEN
        X(1:N)=A(1:N)+B(1:N)
        C(1:N)=X(1:N)*X(1:N)
ELSE IF (N.LEQ.N3) THEN
        DOALL I=1,N
          X(I)=A(I)+B(I)
          C(I)=X(I)*X(I)
        ENDDOALL
     ELSE
        DOALL I=1,P
          X(I:N:P)=A(I:N:P)+B(I:N:P)
          C(I:N:P)=X(I:N:P)*X(I:N:P)
        ENDDOALL
ENDIF
```

当然这里 N_1,N_2,N_3的确定需要经验的积累.

例 2:

```
   DO I=1,N
S:    A(I)=A(I-M)+D(I)
   ENDDO
```

相关性分析可得相关图:

(s) |M|<N

由于|M|≥N 时不存在相关性,可以直接产生向量指令或并行循环. 另外,如果有 M≤0且保证向量操作右边操作数先取,循环仍可向量化,这样引入动态控制技术后,我们就可以得到下列等价的语句:

```
IF ((M.LEQ.0).OR.(M.GEQ.N)) THEN
   A(1:N)=A(1-M:N-M)+D(1:N)
ELSE DO I=1,N
        A(I)=A(I-M)+D(I)
```

```
  ENDDO
ENDIF
```

对于0<M<N的情况，我们还可以通过周期压缩变换成：

```
DO I=1,N,M
   DOALL J=I,I+M-1
         A(J)=A(J-M)+D(J)
   ENDDOALL
ENDDO
```

或通过同步指令实现迭代间的并行执行：

```
DOACR I=1,N
   Wait(I-M)
      A(I)=A(I-M)+D(I)
   sigal(I)
ENDDOACR
```

从而可以产生可能更优但更复杂的带有动态控制的代码. 通过两个简单的例子我们说明了动态控制的作用，它克服了其它优化技术所受到的某些限制，充分利用了运行时的系统和程序特性以提高代码的并行化程度. 然而，我们也必须承认它同样有缺陷，当考虑的因素和状态很多时，它可能导致代码段的极大膨胀.

结束语：构造一个并行化工具是一个难度很大的工作，而构造一个可移植、可扩充、用户界面友好的并行化工具尤其是这样. 在文章中，我们只讨论了基于知识的并行化工具KD-PARPRO的总体结构和启发式转换器的功能描述并以实例介绍了动态控制技术.

并行化工具就是要找到程序和并行处理系统的最佳匹配，以提高运算速度. 即通过工具的改造使得用户程序更适于机器的处理特性，这必然使得工具中直接涉及到了机器特性，若用传统方法编写并行化工具，它的可移植性、可扩充性就要受到限制. 我们在KD-PARPRO中引入知识库概念. 把有关机器特性的知识，基于机器特性的启发信息均以事实、规则的形式存在知识库中，对于不同的机器，只需修改并行化工具KD-PARPRO的知识库，而在增加功能时，只需扩充知识库即可，这使得并行化工具有了很好的可移植性和可扩充性.

参考文献

1 J. R. Allen and K. Kennedy, Automatic Loop Interchange, SIGPLAN Notices 19, 6(1984).

2 U. Banerjee, Speed up of Ordinary Programs, Ph. D. Thesis, 1979.

3 S. R. Midkiff and D. Padua, Compiler Generated Synchronization for Do Loops, ICPP'86.

4 N. J. Nilsson, Principles of Artificial Intelligence, Tioga Publishing Co., 1980.

5 C. D. Polychronopoulos, Parallel Programming and Compilers, Kluwer Aluwer Academic Publisher, 1988.

6 M. J. Wolfe, Optimizing Supercompilers for Supercomputers, Ph. D. Thesis, Oct. 1982.