第 6 卷第 11 期
1995 年 11 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 11
Nov. 1995

一个逻辑程序并行执行的粒度控制模型*

郑宇华　屠红蕾　谢　立　孙钟秀

(南京大学计算机系,南京 210093)

摘要　粒度控制是逻辑程序并行执行的重要问题之一.本文首先引入粒度和粒度值的概念,量化地反映执行一个目标的响应时间,然后建立目标粒度值的计算模型,最后提出了一个并行模型的粒度控制策略.

关键词　粒度控制,粒度计算,并行执行模型,逻辑程序.

在逻辑程序并行执行模型中,一个大的目标往往被分解成若干个小的子目标并行执行,这时,各子目标执行时间的总和可能会增加,但可以缩短此大目标的实际响应时间,从而获取并行效益.通常把顺序执行所需时间较长的目标称为粗粒度目标,而把顺序执行所需时间较短的目标称为细粒度目标.如果一个并行系统对某程序产生了过多的细粒度目标,任务分解开销再加上这些目标之间频繁的通信开销,反而会延长该程序的响应时间;另一方面,如果不对粗粒度目标进行分解,就无法提高并行度,也达不到缩短程序响应时间的目的.所以并行模型中的目标分解必须依据执行效率适当加以控制.

目前已实现了许多逻辑程序并行执行系统,包括 SRI 模型[1],Aurora 模型[2],Muse 模型[3],Reduce－Or 模型[4]以及 PEPSys 模型[5]等,但粒度控制问题还没有得到很好的解决.执行某些程序时,不是产生过多的细粒度目标,就是对可以分解的粗粒度目标不再分解,或者并行目标之间的粒度相差过大,造成各处理机之间负载不平衡,因而降低了系统效率.

为了使系统保持较高的执行效率,本文引入了一个粒度计算模型,以便有效地控制目标分解的粒度大小,从而产生合适数目的并行进程.本文还提出了一个并行系统的粒度控制机制,在该机制下,系统欲分解一个目标时,不仅考虑到由于并行而缩短的响应时间,而且兼顾到由此引起的额外开销.以下部分,首先引入粒度和粒度值的概念,并用它们量化地表示一个目标的响应时间,然后建立粒度计算模型,最后提出并行模型的粒度控制策略.

1　粒　度

1.1　粒度和粒度值

* 本文 1994－05－05 收到,1994－09－08 定稿

作者**郑宇华**,女,1967 年生,博士研究生,主要研究领域为分布式并行处理.**屠红蕾**,女,1971 年生,硕士研究生,主要研究领域为分布式并行处理.**谢立**,1942 年生,教授,主要研究领域为分布式并行处理,智能操作系统,分布式数据库.**孙钟秀**,1936 年生,教授,中国科学院院士,主要研究领域为分布式系统,分布式并行处理.

本文通讯联系人:郑宇华,南京 210093,南京大学计算机系

定义 1. 若 G 为仅包含一个句节的目标语句，则有如下公式成立：

$$SRT(G)=W1_{seq}(G)+SETime(G)+W2_{seq}(G) \tag{1}$$

其中 $SETime(G)$ 表示顺序求解目标 G 所需的时间，$W1_{seq}(G)$ 表示顺序执行目标 G 的等待时间，$W2_{seq}(G)$ 表示等待 G 返回解所需的时间，$SRT(G)$ 表示顺序执行目标 G 的实际响应时间.

定义 2. 若 G 为仅包含一个句节的目标语句，且 G 可以被分解成 n 个并行子目标 g_1，g_2，…，g_n，则有如下公式成立：

$$PRT(G)=W1par(G)+EDT(G)+\max(SETime(g_1),SETime(g_2),\cdots,SETime(g_n))+W2par(G) \tag{2}$$

其中 $W1_{par}(G)$ 表示并行执行 G 的等待时间，$W2_{par}(G)$ 表示并行执行 G 所需的解回收时间，$EDT(G)$ 表示分解 G 所需的额外时间，包括分解目标时间($DT(G)$)、任务转移时间($ST(G)$)、子目标之间的通信时间($CT(G)$)以及结果回收时间($SCT(G)$)，$PRT(G)$ 表示并行执行目标 G 的实际响应时间.

定义 3. 若 G 为仅包含一个句节的目标语句，则有如下公式成立：

$$DSP(G)=SRT(G)-PRT(G) \tag{3}$$

其中 $DSP(G)$ 表示顺序执行目标 G 所需的响应时间与并行执行目标 G 所需的响应时间的差值.

通过对并行分解的合理化分析，可得如下定理：

定理 1. 在一个资源给定的逻辑程序并行执行系统中，若 G 为仅包含一个句节的目标语句，则对 G 的并行分解是合理的，当且仅当：

$$\exists\ g_1,\cdots,g_n \cdot DSP(G)>0$$

并行执行等待时间 $W1_{par}(G)$ 包括了并行调度器所需的通信和同步等额外开销；并行执行解回收时间 $W2_{par}(G)$ 也包括了不同处理器返回解的合成和一致性检查等所需的延迟，因此，$W1_{par}(G)$ 和 $W2_{par}(G)$ 都相应地大于 $W1_{seq}(G)$ 和 $W2_{seq}(G)$，但所增加的时间主要和实现环境有关. 对一个高效的并行系统来说，这部分时间应该远远小于 $SETime(G)$，所以，$DSP(G)$ 的值主要取决于：$SETime(G)$ 和 $EDT(G)+\max(SETime(g_1),SETime(g_2),\cdots,SETime(g_n))$ 之间的差值. 为讨论方便，用 $DSP^*(G)$ 代替 $DSP(G)$ 参与以后的并行分解讨论，且有：

$$DSP^*(G)=SETime(G)-(EDT(G)+\max(SETime(g_1),SETime(g_2),\cdots,SETime(g_n))) \tag{4}$$

由上述分析可知，对 $SETime(G)$ 的估值是确定分解好坏的关键. 于是我们引入粒度来表征一个执行单位的顺序响应时间. 设 p_0 是逻辑程序 P 中具有最短顺序响应时间 t_0 的原子谓词(内部谓词或事实)，定义 p_0 的粒度值为 1；若原子谓词 p 的响应时间为 t，则 p 的粒度值为 t/t_0. 这样根据式(2)，并行执行一个目标的实际响应时间就可以由它的并行子目标的粒度值表示.

1.2 谓词的分类

为了计算逻辑程序中每个谓词的粒度值，首先要对程序进行静态分析，确定谓词之间的关系. 程序中的谓词可分为三类：非递归谓词(NRP)、简单递归谓词(SRP)和复杂递归谓词

(*CRP*).

在逻辑程序谓词集中,用 *Rcall* 关系给出三类谓词定义.

定义 4. 若 f,g 是逻辑程序 P 中的谓词,g 为定义 f 的一个句节,则称 f 调用 g,记作 f *Rcall* g,*Rcall*(P)是满足关系 *Rcall* 的配对集合.

定义 5. 对于逻辑程序 P,定义其闭包关系如下:

$(a,b)\in Rcall(P)$

$(b,c)\in Rcall(P)$

$(a,c)\in Rcall^2(P)$

$Rcall^n(P)=Rcall(P)\cdot Rcall^{n-1}(P)$

$Rcall^+(P)=Rcall^1(P)\cup Rcall^2(P)\cup Rcall^3(P)\cup\cdots\cup Rcall^n(P)$

其中 n 是程序 P 中定义的谓词数.

定义 6. 对于逻辑程序 P,如果 $\forall g\cdot((f,g)\in Rcall^+(P)\Rightarrow(g,g)\notin Rcall^+(P))$ 则 f 是非递归谓词,记作"*NRP*".

一般地,对于谓词 f,若 f 直接或间接调用了 $p_1,p_2,\cdots,p_n$,则:

1. 若存在 i: $1\leqslant i\leqslant n$,且 $p_i=f$,则 f 是递归谓词(即 f 直接或间接调用了自己);

2. 若存在 i: $1\leqslant i\leqslant n$,且 p_i 是递归谓词,则 f 是递归谓词.

否则,f 为非递归谓词.

定义 7. 对于逻辑程序 P,如果:

$((f,f)\in Rcall(P))\wedge\forall g\cdot(((f,g)\in Rcall^+(P)\wedge f\neq g)\Rightarrow(g,g)\notin Rcall^+(P))$

则 f 是简单递归谓词,记作"*SRP*".

一般地,对于谓词 f,若 f 直接或间接调用了 $p_1,p_2,\cdots,p_n$,则 f 是简单递归谓词,当且仅当:

1. 存在 i: $1\leqslant i\leqslant n$,且 $p_i=f$(即 f 直接或间接调用了自己);

2. 对任一 i: $1\leqslant i\leqslant n$,若 $p_i\neq f$,则 p_i 是非递归谓词

(即 f 不再直接或间接调用 f 以外的递归谓词).

定义 8. 对于逻辑程序 P,如果:

$$\exists g\cdot(((f,g)\in Rcall^+(P))\wedge(f\neq g)\wedge((g,g)\in Rcall^+(P)))$$

则 f 是复杂递归谓词,记作"*CRP*".

一般地,对于谓词 f,若 f 直接或间接调用了 $p_1,p_2,\ldots,p_n$,则 f 是复杂递归谓词,当且仅当存在 i:$1\leqslant i\leqslant n\cdot p_i\neq f$ 且 p_i 是递归谓词即 f 直接或间接调用了除自己以外的递归谓词.

2　粒度计算模型

2.1　结　构

图 1 给出了一个粒度计算模型的结构.

逻辑程序 P 的谓词粒度计算过程可表述如下:

typedef struct{

```
    char Callname[NAME_LENGTH];
    char Calledname[NAME_LENGTH];
    int Callnum;
}MPR;
MPR Mpr[MAX_PREDICATE_NUM][MAX_PREDICATE_NUM];
MPR Mpr+[MAX_PREDICATE_NUM][MAX_PREDICATE_NUM];
```

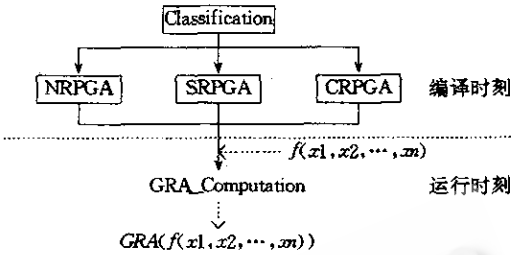

图1 粒度计算模型结构

编译时刻：

(1)设 P 中谓词为 $p_1,p_2,\cdots,p_n,n$ 为谓词数目，则对 $1\leqslant i,j\leqslant n$，有：

$Mpr[i,j].Callname=p_i$

$Mpr[i,j].Calledname=p_j$

$$Mpr[i,j].Callnum=\begin{cases}p_i \text{ 调用 } p_j \text{ 的次数} & if\ p_iRcall\ p_j\\ 0 & otherwise\end{cases}$$

(2)求得 Mpr 的传递闭包 Mpr^+；

(3)$PROC\_CALL\ Classification()$；

调用过程 $Classification()$，将逻辑程序中出现的谓词按上节定义的 3 种类型分类.

(4)$PROC\_CALL\ NRPGA()$；

调用过程 $NRPGA()$，计算所有非递归谓词(NRP)的粒度值.

(5)$PROC\_CALL\ SRPGA()$；

调用过程 $SRPGA()$，求得所有简单递归谓词(SRP)的粒度值计算表达式.

(6)$PROC\_CALL\ CRPGA()$；

调用过程 $CRPGA()$，求得所有复杂递归谓词(CRP)的粒度值计算表达式.

运行时刻：

$GRA\_Computation(f(x_1,\cdots,x_n))$；

返回 f 的粒度值，f 是运行时刻收到的要求粒度值的谓词.

以下我们给出各模块的具体算法描述.

2.2 分类算法

分类算法基于 Mpr 和 Mpr^+ 将程序 P 中的所有谓词分为 3 类，结果保存在数组 $TAGList$ 中.

主要过程描述如下：

```
typedef struct{
  char name[NAME_LENGTH];
  int tag;
}TAG;
TAG TAGlist[MAX_PREDICATE_NUM];
```

设 P 中谓词为 $p_1,p_2,\cdots,p_n,n$ 为谓词数目，则对$1\leqslant i\leqslant n$，有：

$TAGlist[i].name=p_i$

$$TAGlist[i].tag=\begin{cases}CRP & if\ \exists\ j,1\leqslant j\leqslant n,j\neq i,Mpr[i,j].Callnum\neq 0\\ & \quad \wedge Mpr^{+}[j,j].Callnum\neq 0\\ SRP & if\ Mpr[i,i].Callnum\neq 0\ and\ \forall\ j,1\leqslant j\leqslant n,\\ & \quad Mpr[i,j].Callnum\neq 0\rightarrow Mpr^{+}[j,j].Callnum==0\\ NRP & otherwise\end{cases}$$

2.3 NRPGA 算法

对于 *NRP* 的粒度计算,首先得到仅由事实或内部谓词组成的基本谓词的粒度值,不妨设事实的粒度值为1,内部谓词的粒度值根据其操作的执行时间定为一个常量.然后定义 *NRP* 的粒度值为它的所有或并行和与并行子目标的粒度和.表示为:

$$Grain(f)=Grain(f_1)+Grain(f_2)+\cdots+Grain(f_n) \tag{5}$$

$f_1,f_2,\cdots,f_n$ 是 f 的多个或并行目标.

$$Grain(f_i)=Grain(f_{i_1})+Grain(f_{i_2})+\cdots+Grain(f_{i_m}) \tag{6}$$

$f_{i_1},f_{i_2},\cdots,f_{i_m}$ 是目标 f_i 的多个与并行目标.

重复使用这两个公式,将 *NRP* 的粒度值用它的或并行和与并行子目标的粒度值来代替,直到它被简化为事实和内部谓词的粒度和,最后将这些事实和内部谓词的粒度值相加得到该 *NRP* 的粒度值.所有的粒度计算结果存入到 *NRPlist* 中.*NRP* 的粒度计算过程表示如下:

```
typedef struct{
  char name[NAME_LENGTH];
  int Gvalue;
  int Callnum;
}NRP;
NRP NRPlist[MAX_PREDICATE_NUM];
int NRP_number=0,Count=0;
```

设 P 中谓词为 $p_1,p_2,\cdots,p_n$,n 为谓词数目,计算步骤如下:

(1)对 $1\leqslant i\leqslant n$,若 $TAGlist[i].tag==NRP$,则有:

```
NRP_number=NRP_number+1;
NRPlist[NRP_number].name=TAGlist[i].name;
```

$$NRPlist[NRP\_number].Callnum=\sum_{j=1}^{n}Mpr[i,j].Callnum$$

$$NRPlist[NRP\_number].Gvalue=\begin{cases}1\text{或常量} & if\ NRPlist[NRP\_number].\\ & \quad Callnum==0\\ 0 & otherwise\end{cases}$$

(2)对 $1\leqslant i\leqslant NRP\_number$,若 $NRPlist[i].Callnum==0$,则有:

```
Count=Count+1;/* 已处理过的 NRP 数目加1 */
NRPlist[j].Gvalue=NRPlist[j].Gvalue+Mpr[j1,i1].Callnum
                  * NRPlist[i].Gvalue;
```

$NRPlist[j].Callnum=NRPlist[j].Callnum-Mpr[j_1,i_1].Callnum$

其中$1\leqslant j,j_1,i_1\leqslant NRP_number$，$Mpr[j_1,i_1].Callname==NRPlist[j].name$，

$Mpr[j_1,i_1].Calledname==NRPlist[i].name$.

(3)若$Count<NRP_number$，则转(2).

2.4 SRPGA 算法

由于 *SRP* 谓词仅调用其它 *NRP* 谓词和自身，并且 *NRP* 的粒度值已在 *NRPGA* 中求得，所以计算 *SRP* 谓词的粒度时，应先将它所调用的 *NRP* 谓词的粒度值相加，假设为常数 *B*，则 *SRP* 的粒度可用公式(7)表示：

$$G(n)=\begin{cases}1 & if\ n=0\\ A*G(n-1)+B & otherwise\end{cases} \tag{7}$$

其中 *A* 表示在多个定义子句中 *SRP* 调用自身的次数. *n* 表示 *SRP* 的递归因子，$n=0,1,\cdots$. 然后重复迭代操作得到：

$$G(n)=\frac{A^n(A+B-1)}{A-1}+\frac{B}{1-A} \qquad if\ A\neq 1 \tag{8}$$

$$G(n)=1+nB \qquad if\ A=1 \tag{9}$$

SRPGA 的计算结果将得到一个形如公式(8)或(9)的公式，并存于 *SRPlist* 中. 该算法描述如下：

```
typedef struct{
  int factor;
  int A;
  int B;
}FORMULAR;
typedef struct{
  char name[NAME_LENGTH];
  FORMULA Gvalue_formula;
}SRP;
SRP SRPlist[MAX_PREDICATE_NUM];
int SRP_number=0;
```

设 *P* 中谓词为 $p_1,p_2,\cdots,p_n$，*n* 为谓词数目，对$1\leqslant i\leqslant n$，若 $TAGlist[i].tag==SRP$，则有：

$SRP\_number=NRP\_number+1$；$SRPlist[SRP\_number].name=TAGlist[i].name$；

$SRPlist[SRP\_number].Gvalue\_formula.A=Mpr[i,i].Callnum$；

$SRPlist[SRP\_number].Gvalue\_formula.B=p_i$ 所调用的 *NRP* 的粒度和；

$SRPlist[SRP\_number].Gvalue\_formula.factor$ 运行时刻确定；

2.5 CRPGA 算法

CRP 谓词的粒度计算与其它两类相比更为复杂，由于 *CRP* 谓词不仅可能调用其它 *NRP* 和 *SRP*，而且可能调用其它 *CRP*，为了简化计算过程，这里仅讨论受下列限制的 *CRP*：

·*CRP* 仅调用自身,不调用其它 *CRP*;

·仅调用一种 *SRP*,且 *SRP* 的递归因子与 *CRP* 的递归因子呈线性关系,该关系用 $p(n)$表示.

因为 *SRP* 的粒度计算公式已在前面的 *SRPGA* 中获得,我们可以给出 *CRP* 的计算公式如下:

$$G(n)=\begin{cases}1 & if\ n=0\\ G(n-1)+G_{SRP}(p(n))+B_0 & otherwise\end{cases} \tag{10}$$

其中 $G_{SRP}(p(n))$代表 *SRP* 的粒度,多数情况下 $p(n)=n$,因此我们在公式(10)中用 n 代替 $p(n)$. B_0表示所调用的 *NRP* 的粒度值总和. 重复迭代操作,可以得到:

$$G(n)=1+n\left(1+\frac{nB(2-A)}{1-A}+\frac{A(A+B-1)(A^n-1)}{(A-1)^2}+\frac{nB}{1-A}\right)+B_0n \qquad A\neq 1 \tag{11}$$

$$G(n)=n+\frac{nB(n+1)}{2} \qquad (A=1) \tag{12}$$

公式(11)和(12)将是我们在算法 *CRPGA* 结束时为每一个 *CRP* 提供的计算式. 计算过程如下:

```
typedef struct{
  char name[NAME_LENGTH];
  int B0;
  FORMULA Gvalue_formula;
}CRP;
CRP CRPlist[MAX_PREDICATE_NUM];
int CRP_number=0;
```

设 *P* 中谓词为 $p_1,p_2,\cdots,p_n,n$ 为谓词数目,对$1\leqslant i\leqslant n$,若 *TAGlist*[*i*].*tag*==*CRP*,则有:*CRP_number*=*CRP_number*+1;

CRPlist[*CRP_number*].*name*=*TAGlist*[*i*].*name*;

设 *j* 满足 *Mpr*[*i*,*j*].*Callnum*≠0且 *Mpr*[*i*,*j*].*Calledname*≠*Mpr*[*i*,*j*].*Callname*,

若*Mpr*$^+$[*j*,*j*].*Callnum*≠0,则

CRPlist[*CRP_number*].*Gvalue_formula*=*SRPlist*[j_1].*Gvalue_formula*;

其中 j_1满足 *SRPlist*[j_1].*name*==*Mpr*[*i*,*j*].*Calledname*.

否则

*B*0=*B*0+*NRPlist*[j_1].*Gvalue*;

其中 j_1满足 *NRPlist*[j_1].*name*==*Mpr*[*i*,*j*].*Calledname*.

2.6 GRA_Computation 算法

GRA_Computation 模块的功能在运行时刻完成,它根据前面各步获得的信息,为谓词确定粒度值.

该算法描述如下:

Pre_name(*predicate*):返回谓词名.

Pre_para(*predicate*):返回谓词的递归因子,它出现在第一个参数位置,或由用户显式

标注.

$char\ name[NAME\_LENGTH]$;

$int\ factor$;

$f(X_1,X_2,\cdots,X_n)$为$GRA_Computation()$收到的欲求粒度值的谓词,

$name=Pre\_name(f(X_1,X_2,\cdots,X_n))$;

$factor=Pre\_para(f(X_1,X_2,\cdots,X_n))$;

设i满足$TAGlist[i].name==name$,则有:

(1)若$TAGlist[i].tag==NRP$,则返回$NRPlist[j].Gvalue$,其中j满足$NRPlist[j].name==name$;

(2)若$TAGlist[i].tag==SRP$,则返回

$$\begin{cases} A^n*(A+B-1)/(A-1)+B/(1-A) & if\ A\neq 1 \\ 1+n*B & otherwise \end{cases}$$

其中$A=SRPlist[j].Gvalue_formula.A$,$B=SRPlist[j].Gvalue\_formula.B$,$n=factor$,$SRPlist[j].Gvalue\_formula.factor=factor$,而$j$满足$SRPlist[j].name==name$;

(3)若$TAGlist[i].tag==CRP$,则返回

$$\begin{cases} \begin{cases} 1+n*(1+n*B*((2-A)/(1-A)) \\ +(A+B-1)/(A-1)*(A*(A^n-1)/(A-1)) \\ +n*B/(1-A))+B_0*n \end{cases} & if\ A\neq 1 \\ n+B*n*(n+1)/2 & otherwise \end{cases}$$

其中$A=CRPlist[j].Gvalue_formula.A$,$B=CRPlist[j].Gvalue\_formula.B$,$n=factor$,$CRPlist[j].Gvalue\_formula.factor=factor$,而$j$满足$CRPlist[j].name==name$;

3 粒度控制策略

在一个并行执行模型中,那些不再被分解的目标将顺序执行,所以执行一个目标的响应时间可以用它的并行子目标的顺序响应时间来表示.也就是说,可以用它的并行子目标的粒度值来表示.这样基于对目标谓词的粒度计算模型及应用定理1就可得到如下分解合理性判定式:

$$\begin{aligned} &GV(G)-(EDT(G)/t_0+\max(GV(g_1),GV(g_2),\cdots,GV(g_n)))>0 \\ \Rightarrow &GV(G)-((DT(G)+ST(G)+CT(G)+SCT(G))/t_0 \\ &+\max(GV(g_1),GV(g_2),\cdots,GV(g_n)))>0 \end{aligned} \tag{13}$$

其中$GV(x),(x=G,g_1,\cdots,g_n)$代表了目标$x$的粒度值($GranularityValue$);$DT$,$ST$,$CT$,$SCT$的含义参见定义2.

根据公式(13),在确定一个目标是否适合分解前,我们需要考虑以下几个因素:

首先,在并行模型中执行一个程序要预先估计各目标的顺序响应时间,也就是说要给出它们的粒度值.我们在前面已讨论过计算方法.

再来考虑公式(13)中的$DT(G)+ST(G)+CT(G)+SCT(G)$.一个特定目标的分解、转移、通信和结果回收时间不能在编译时刻确定下来,即使在运行时刻也是各不相同的,它

随着一个目标分解出的子目标个数,目标的解数目以及转移任务数的不同而不同.分析的结果发现这种时间差异在同一个模型中相差不大,不妨设其平均值为 AS,而在实际应用模型中,用 $\alpha * AS$ 代替 AS.如果 $GV(G)>\max(GV(g_1),GV(g_2),\cdots,GV(g_n))+\alpha * AS$,则认为 G 是适合分解的;否则 G 将顺序执行.这里 α 是一个调节因子,根据动态执行条件,可以适当调节 α,使得机制能够自适应.在确定 α 时,要考虑几个因素,例如可获得的处理机个数、这些处理机当前的负载状况、要求执行的程序长度等.如果系统处理机个数较少,或者我们希望目标粒度较粗,则可以将 α 取一个较大的值;如果系统处理机个数较多,或者我们希望目标粒度较细,则 α 可以取一个较小的值.

通过选取适当的 α,我们可以调节分解程度以达到较为理想的并行效率.

4　与其它粒度控制模型的比较

目前已被提出的一些粒度控制算法有任务窃取算法[6],全局负载算法[7]和 *CG* 算法[8]等.任务窃取算法的主要思想是让空闲处理机去获取非空闲处理机的任务,以达到充分利用处理机资源的目的.全局负载算法要求,系统负载不得低于某个阈值,一旦低于该值,就分解产生任务.在 *CG* 算法中,把一个操作多于 m 的目标定为大粒度目标,对它进行任务分解,并将分解出的任务移至其它处理机执行.

任务窃取算法可能达到一个较高的并行度,但这种不受限制的并行执行也可能会使通信代价急剧增长,从而降低了整个系统的效率.在我们的机制中,引入了一个阈值来控制分解程度,期望在提高的并行度和增加的通信开销之间保持平衡,以改善系统的性能.

全局负载算法是利用系统产生的任务数做为阈值,来调节全局系统负载的.在这个机制中,当系统想分解一个任务时,它并不根据任务粒度的大小,而是任选一个任务分解,这可能会产生过细粒度的任务,而使多个处理机之间实际负载不均.在我们的模型中,系统有意选取粗粒度的任务来分解,这可以避免产生过多的细粒度目标.

至于 *CG* 算法,它不仅考虑到了任务的粒度,而且设置了一个粒度阈值来控制分解,但是任务分解是在 m 个操作完成后进行的,这可能会降低并行度.在我们的模型中,根据已有的粒度值,能够在目标执行前决定是否对它进行分解,以此提高并行度.

5　小　结

本文提出了一个粒度控制模型,该模型包含一个粒度计算模型和粒度自适应控制机制,以支持逻辑程序的并行执行.它为程序中每个谓词提供一个粒度值,如果该值大于预定的阈值,则目标是适合分解的,否则目标将顺序执行.

基于本文提出的粒度计算模型,我们作了初步实现,并对一些简单的程序实例作了一定的分析,效果良好.但对一些较为复杂的程序,还不能够彻底分析,原因在于对复杂递归谓词的处理能力还有限.今后将在这方面作进一步的努力.

参考文献

1　Warren D H D. The SRI—model for OR—parallel execution of Prolog abstract design and implementation. Proc.

of the IEEE Fourth Symposium on Logic Programming, San Francisco, 1987.

2 Lusk E *et al*. The aurora Or—parallel Prolog system. New Generation Computing 7, 1990.

3 Khayri A M Ali, Karlsson R. The muse OR—parallel Prolog model and its performance. Proc. of the North American Conf. on Logic Programming, MIT Press, October 1990. 757—776.

4 Kale L V. The REDUCE—OR process model for parallel evaluation of logic program. Logic Programming, Proceedings of the Fourth International Conference, 1987.

5 Westphal H, Robert P, Chassin J *et al*. The PEPSys model: combining backtracking, AND— and OR— parallelism. Proc. of the IEEE Fourth Symposium on Logic Programming, San Francisco, 1987.

6 Khayri A M Ali, Karlsson B. The muse approach to Or—parallel Prolog. International Journal of Parallel Programming, 1990, **19**(2).

7 Shapiro E. An Or—parallel execution algorithm for prolog and its FCP implementation. Logic Programming, Proceedings of the Fourth International Conference, 1988.

8 Barak A. Granularity control of fine—grained parallel computation. Lecture Notes, Given in Computer Science Department of Nanjing University, 1993.

A GRANULARITY CONTROL MODEL FOR PARALLEL EXECUTION OF LOGIC PROGRAMS

Zheng Yuhua Tu Honglei Xie Li Sun Zhongxiu

(*Department of Computer Science, Nanjing University, Nanjing* 210093)

Abstract Granularity control is a main problem of parallel execution of logic programs. This paper first introduces the concepts of granularity and granularity value to numerically reflect the response time of executing a goal. Then it constructs a computation model to estimate the granularity value of each goal. The control strategy of granularity in the parallel model is also presented.

Key words Granularity control, granularity computation, parallel execution model, logic program.