语义约束 Constraintl，并且可以动态地和有向链 Link1进行捆绑，或者释放捆绑：

```
Meta_Constraint * Constraintl;
Constraintl=new Meta_Constraint(DirectedLink);
Constraintl->AddAttribute(a1,a2,a3);
Link1. BindConstraint(Constraintl);
Link1. UnBindConstraint(Constraint1);
```

这种机制充分体现了引擎的可剪裁性。

实例对象。当设计者对引擎进行剪裁，或者用户编辑超文档的时候，引擎会创建各种实例对象，它们和元对象有着被描述的关系。引擎提供了以下的实例对象类，它们和元对象类一一对应：

- Anchor_Class。负责创建所有的锚实例对象。
- Link_Class。负责创建所有的链实例对象。
- Node_Class。负责创建所有的结点实例对象。
- Document_Class，负责创建所有的文档实例对象。
- Constraint_Class，负责创建所有的语义约束实例对象。
- Pspec_Class，负责创建所有的展示实例对象。
- Service_Class。负责创建所有的服务实例对象。
- Perspective_Class。负责创建所有的视图实例对象。

当引擎创建实例对象时，从元对象中，它可以理解 Viewer 发过来的数据，从而在 Shore 数据库中建立真正的持久对象。如，Link_Class * lpAl = new Link_Class(T_IsBaseOn)；T_IsBaseOn 是设计者定义的链元对象，描述了实际的链 lpAl 的属性格式和语义约束规则。

实例对象组成的有机网络在元对象上的映射视图，称之为元视图。它表达了元对象的属性和它们之间的关系。通过对元视图的操作，可以增删元节点，修改锚的属性，定义链，捆绑新的语义约束，并且这些操作也反映在实例视图上。引进元视图的概念，使得对引擎进行可视化剪裁成为可能。

（下转第93页）

（上接第116页）

```
      S=0
      DO 130 K=1,10
      DO 140 M=1,10
      S=S+1.0D+00
      CALL F_AUTOPAR_LOOP_SPLIT (8.2, F_AU-
     TOPAR_LOOP_START, & F_AUTOPAR_LOOP_
     END)
      DO 150 J = F_AUTOPAR_LOOP_START, F_AU-
     TOPAR_LOOP_END
      DO 160 I=1,10
      A(I,J,K,M)=1010.0+j
      B(J,I,K,M)=20.0+j
160 CONTINUE
150 CONTINUE
140 CONTINUE
130 CONTINUE
      CALL f_autopar_set_range_d(3,1,10)
      CALL f_autopar_set_range_ud(2,0,0)
      CALL f_autopar_set_range_d(1,1,10)
      CALL f_autopar_set_range_d(0,1,10)
      CALL f_autopar_layout(a)
      CALL f_autopar_collect(a)
      CALL f_autopar_set_range_ud(3,0,0)
      CALL f_autopar_set_range_d(2,1,10)
      CALL f_autopar_set_range_d(1,1,10)
      CALL f_autopar_set_range_d(0,1,10)
      CALL f_autopar_layout(b)
      CALL f_autopar_collect(b)
      CALL F_AUTOPAR_OVER
      END
```

很明显，该程序并行的是150循环。在这里，读者应该特别注意的是那些小写字母表示的 AUTOPAR 函数调用语句，它们本应该在150 CONTINUE 和140 CONTINUE 之间生成，但由于数组 A，B 都没在循环130和140中被引用，因此经过通讯优化分析，它们被放到了130循环之外。

结论 我们简要地介绍了曙光2000上的并行库和并行识别工具 AUTOPAR。在经过对大量 BENCHMARK 程序(NASA，PERFECT 等)的实际测试后，我们发现尽管 AUTOPAR 能够并行大部分 DO 循环，但实际的运行效率并不高，主要瓶颈在于数据收集函数 COLLECT(通信)。因此，该系统今后还应该在并行库优化和通信优化上下工夫。虽然如此，该系统对一些通讯少的应用，还是能得到很好的并行效果。如对三维迭前深度偏移(克希霍夫方法) mig. f 就能使加速比大体呈线性。

参 考 文 献

[1] 吉敏阳，并行重构系统后端[硕士学位论文]，1994

[2] 张兆庆、乔如良，利用超级编译技术优化串行程序，软件学报，1995年增刊

[3] 吉晓梅等，含数组引用的过程间数据流分析，软件学报，1995年增刊

[4] 高念书等，实用数据依赖分析方法，计算机学报，1995年4月

[5] 张兆庆、乔如良，PORT：并行优化重构工具集，计算机学报，1994年12月

[6] 刘任，MPP 机上代码生成技术，硕士论文，1995

[7] ASPAR—An Automatic Parallelizer for C Programs，1991

计算机科学1998Vol.25№5

一个自动并行程序转换工具

An Automatic Transformation Tool for Parallel Programs

胡永刚 祝 明

(中国科学院计算所智能计算机研究开发中心 高性能计算机研究中心 北京100080)

摘 要 This paper studies an automatic parallel analysis tool AUTOPAR at Dawning-2000. It transforms F77 programs to parallel programs based on parallel communication library PVM. AUTOPAR includes parallel communication library and parallel compiler. The parallel communication library comes from Aspar(another parallel tool based on Express communication library),but the communication function is different from it. The compiler comes from PORT[5]. It can not only generate parallel DO loops,but also consider the problem of communication optimization.

关键词 Automatic parallel,Parallel communication library,PVM,Data distribution,Loop split

1. 引言

在数值计算中,往往循环迭代所占的时间最多,因此,我们的并行库和并行识别器 AUTOPAR 是针对循环实现的。DO 循环经过 PORT 系统的分析(流分析、依赖分析和并行分析),我们就知道它在哪层是可并行的以及是否存在体间依赖等。对于既可并行,又无体间依赖的循环,AUTOPAR 首先去判断它是否可转换成并行库调用。然后,分割可转换的并行循环,把它们分配在不同的处理机上,即每台处理机处理部分循环。这就实现了该循环的并行执行,大大降低了循环的执行时间,但也导致每台机器上只有部分数据是有效的。因此,在循环的出口处,要进行数据收集工作,这就反过来影响了并行的效率。我们的工作正是基于这一考虑,从两方面来减少并行的开销。一是数据收集函数的设计实现,该函数通过主从机数据传输、数据块移动等技术,来提高函数的运行效率。二是代码生成时的通信优化,它把在循环体内的数据收集函数放在循环之外,以减少收集函数的运行次数。

2. 并行库函数

曙光2000并行库 AUTOPAR 的设计参考了 Aspar 并行库,不同之处在于 Aspar 的底层库是 Express 并行环境,而 AUTOPAR 则是基于 PVM 并行环境的。该库函数适合实现满足如下条件的并行循环:

①循环控制变量增量值为1。②在所有下标表达式中,涉及到循环控制变量的表达式,其循环控制变量的系数必须为1。下面分别给出这些库函数的详细描述。

2.1 相关 PVM 环境的库函数

这类函数用于 PVM 同 AUTOPAR 并行库函数的信息交换。

● F_AUTOPAR_INITIAL(程序名)。得到当前 PVM 的一些环境变量(当前节点的 tid 号,PVM 的节点数等),该函数在应用程序的最开始调用一次。

● F_AUTOPAR_OVER()。终止该程序在 PVM 中的运行。在这里设一个障碍点,等待所有节点程序运行到此。该函数在应用程序结束前调用一次。

2.2 相关数组声明的库函数

这类函数用于把每个数组的信息(大小,类型)通知 AUTOPAR 库函数。

● F_AUTOPAR_SET_SIZE (array_dim, start,size),设置一个数组在第 array_dim 维上的起始点 start 和长度 size。在这里要特别提醒读者注意的是,我们的维号 array_dim 与 F77程序的维号顺序相反。

● F_AUTOPAR_SET_DECL (address_array, array_ndim, type_of_element)。声明数组 address_array 的维数为 array_ndim,而它的类型为 type_of_element。它要紧跟在一个或几个(由维数

决定)F_AUTOPAR_SET_SIZE()函数之后。

下面是这两个函数的使用方法。

假设 F77程序中有个声明为：

```
REAL*8A(100,2:200,3:300)
```

则 AUTOPAR 对 A 的声明相应为：

```
F_AUTOPAR_SET_SIZE(2,1,100)/*F77中的第一维，我们以2来表示*/
F_AUTOPAR_SET_SIZE(1,2,199)/*F77中的第二维，我们以1来表示.*/
F_AUTOPAR_SET_SIZE(0,3,298)/*F77中的第三维，我们以0来表示.*/
F_AUTOPAR_SET_DECL(A,3,F_DOUBLE)
```

其中 F_DOUBLE 表明数组 A 的类型为双精度。

2.3 相关循环的库函数

相关循环的库函数才是 AUTOPAR 库函数的一类关键函数。它们把一个循环分布到所有的 PVM 处理机上，每个处理机计算部分数据，在循环结束后再综合各个处理机的计算结果，使每个处理机都保留最新的计算结果。

● F_AUTOPAR_LOOP_SPLIT (max, min, l_start,l_end)。把由[min,max]标记的循环迭代空间分布到 PVM 的各个处理机上。[l_start,l_end]分别标记该循环在当前处理机上的迭代空间。

● F_AUTOPAR_SET_RANGE_D (array_dim,start,end)。设置被收集数组在 array_dim 维上的分布区间[start,end]。该函数用于设置数组下标不含并行循环控制变量的数组维。

● F_AUTOPAR_SET_RANGE_UD(array_dim,left,right)。设置被收集数组在 array_dim 维上的分布区间[l_start+left,l_end+right]。该函数用于设置数组下标含并行循环控制变量的数组维。

● F_AUTOPAR_LAYOUT(address_array)。该函数要紧随 F_AUTOPAR_SET_RANGE_D, F_AUTOPAR_SET_RANGE_UD 之后，用于存储数组 address_array 在当前处理机上的有效空间信息，为函数 F_AUTOPAR_COLLECT 服务。

● F_AUTOPAR_DIST(address_array)。该函数要紧随 F_AUTOPAR_SET_RANGE_D, F_AUTOPAR_SET_RANGE_UD 之后，用于处理循环中定义—使用型数组，存储该数组 address_array 在当前处理机上的有效空间信息，为函数 F_AUTOPAR_COLLECT 服务。

● F_AUTOPAR_COLLECT (address_array)。之所以要引进该函数，是因为一个循环是在多个处理机上分别运行的，这就造成了每个处理机的数据(数组)局部性。该函数利用处理机间的通讯使数组 address_array 在各个处理机上的值都相同。由于涉及到大量通讯，该函数常常会成为发挥并行效率的瓶颈。

下面是这六个函数的使用方法。

假设原 F77循环为：

```
do 10 j=200,300
   do 10 i=1,400,1
   a(i,j)=10000.0+10*i+j
   b(j,i)=10000.0+j+10*i
10 continue
```

则并行循环代码为：

```
CALL F_AUTOPAR_LOOP_SPLIT(300,200,F_AUTOPAR_LOOP_START,F_AUTOPAR_LOOP_END)
DO2230J = F_AUTOPAR_LOOP_START, F_AUTOPAR_LOOP_END
    DO2240I=1,400
    A(I,J)=I*10+10000.0+J
    B(J,I)=J+10000.0+I*10
    2240 CONTINUE
2230 CONTINUE
CALL F_AUTOPAR_SET_RANGE_D(1,1,400)
CALL F_AUTOPAR_SET_RANGE_UD(0,0,0)
CALL F_AUTOPAR_LAYOUT(A)
CALL F_AUTOPAR_COLLECT(A)
CALL F_AUTOPAR_SET_RANGE_UD(1,0,0)
CALL F_AUTOPAR_SET_RANGE_D(0,1,400)
CALL F_AUTOPAR_LAYOUT(B)
CALL F_AUTOPAR_COLLECT(B)
```

3. 并行识别器

上一节我们给出了曙光2000上的并行库函数，现在介绍我们的并行转换工具 AUTOPAR。AUTOPAR 是从源源变换并行开发工具 PORT[5]发展而来，它利用 PORT 系统的前端信息、控制流/数据流信息、依赖图信息以及并行化信息来进行并行 DO 循环到并行库调用的转换，从而生成运行在曙光2000上的并行程序。

在 AUTOPAR 中主要要完成五方面的工作。①在程序开始和结束时生成相关 PVM 环境的库函数调用。②对程序中每个数组生成相关数组声明的库函数调用。③对并行循环进行分割，把一个并行循环分配到多个处理机上运行，实现方法是在并行循环前生成库函数 F_AUTOPAR_LOOP_SPLIT 的调用语句。④对循环中定义或定义-使用出现的数组进行数据分割和收集，分别生成函数 F_AUTOPAR_SET_RANGE_D, F_AUTOPAR_SET_RANGE_U, F_AUTOPAR_LAYOUT, F_AUTOPAR_DIST, F_AUTOPAR_COLLECT 的调用语句。⑤进行通讯优化工作，假设我们对循环 L 中的并行循环 L1按上述规则生成了并行库的调用，并且对 L1中定义性出现的数组 A 进行数据收集，若数组 A 在 L 中的所有出现均在 L1循环体内，则此时我们可以将数组 A 的收集工作移到 L 之后完成。下面是算法的基本流程，其中涉及到的一些术语，请参见[1][3][4]。

算法:AUTOPAR 并行库代码生成算法

输入:子程序或函数 S。

输出:等价于 S 的并行代码。

```
/*info0,info1,info2:一组数据结构*/
/*outLoops:当前循环的所有外层循环*/
/*stat.语句*/
/*无体间依赖:每个循环迭代都不会用到其它循环迭代产生的数据*/
codeGeneration(S)
{
   for(所有 L∈S){
       outLoops=NULL;
       info0=info0+processLoop(L);
   }/*endfor*/
   /*生成函数 S 的头语句和说明语句*/
   /*根据 info1生成由函数 F_AUTOPAR_SET_SIZE,
   F_AUTOPAR_SET_DECL 组成的 AUTOPAR 数组
   说明语句*/
   for(所有 S 中的可执行语句 stat){
       if(stat 是 DO 语句)
           /*根据 info0生成并行/串行 DO 循环*/
       else
           /*生成该语句*/
   }
}
processLoop(L)
{
   if(L 是并行循环,而且无体间依赖){
           info1=/*记录循环控制信息和循环体中需要
           分割和收集的数组信息*/
           /*以下语句先生成到一个临时文件 tmp.L*/
           /*生成循环分割函数 F_AUTOPAR_
           LOOP_SPLIT(上界,下界,L,U)*/
           /*需要分割的数组(在循环中定义-使用出
           现),对每一个生成由函数 F_AUTOPAR_
           SET_RANGE_D, F_AUTOPAR_SET_
           RANGE_UD,F_AUTOPAR_DIST 组成的
           语句序列*/
           /*生成 DO I_L=L,U*/
           /*生成循环体*/
           /*生成 ENDDO*/
           /*处理需要收集的数组 CA(在循环中只有定
           义出现)*/
           for(对每一个 A in CA){
           /*生成由函数 F_AUTOPAR_SET_
               RANGE_D, F_AUTOPAR_SET_
               RANGE_UD,F_AUTOPAR_LAYOUT
               组成的语句序列*/
           if(A 在 outLoops 中要使用)
               /*生成 F_AUTOPAR_COLLECT 函
               数*/
           else
               info2=/*记录该 F_AUTOPAR_COL_
               LECT 函数将生成在哪层外的信息*/
           }/*endfor*/
           info0=info0+rmp.L
   }/*endif*/
   else{/*该循环是串行的,或体间有依赖*/
           outLoops=outLoops-{L};
           /*生成 DO I_L=下界,上界*/
           for(所有在 L 中的语句 stat){
               if(stat 是 DO 语句){
                   info0=info0+processLoop(stat);
               }
               else
                   /*生成该语句*/
           }
           /*生成 ENDDOI_L 语句*/
           /*根据 info2,生成相应 F_AUTOPAR_COL_
           LECT 语句*/
   }
}
```

4. 一个测试结果

上面我们给出了 AUTOPAR 的并行库和并行算法,这里我们给出一个完整的 F77例子,来看看 AUTOPAR 生成的并行代码。

例:

```
program f4
implicit real*8(a-h,o-z)
dimension A(10,10,10,10),b(10,10,10,10)
integer i,j,k,m
data a/10000*10.0/
s=0
do 10 k=1,10,1
     do 10 m=1,10,1
       s=s+1
       do 10 j=2,8
         do 5 i=1,10,1
           a(i,j,k,m)=1010.0+j
           b(j,i,k,m)=20.0+j
5        continue
10 continue
   stop
   end
```

在经过 AUTOPAR 的分析后,该程序转换成了能在曙光2000上运行的并行程序,其中的 GETARG 函数的功能是读出执行文件名,它与并行库无关。

```
      PROGRAM F4
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10,10,10),B(10,10,10,10)
      INTEGER I,J,K,M
      DATA A/10000*10.0/
C----------AUTOPAR SYSTEM VARIABLES-----
C---------1. THE AUTOPAR ENVIROMENT VARI-
     ABLE-------
      INTEGER F_AUTOPAR_LOOP_START
      INTEGER F_AUTOPAR_LOOP_END
C DECL_TYPE:CHARACTER=0,LOGICAL=1,IN-
TEGER=2,REAL=3,DOUBLE=4,
C COMPLEX=5
      COMMON/AUTOPAR/F_AUTOPAR_LOOP_
      START,F_AUTOPAR_LOOP_END
      INTEGER INOCHNG
      DATA INOCHNG/-10/
C-------------------------
C----------SYSTEM INITIAL----------
      CHARACTER*10   F_AUTOPAR_PROGRAM_
           NAME
      CALL GETARG(0,F_AUTOPAR_PROGRAM_
           NAME)
      CALL F_AUTOPAR_INITIAL(F_AUTOPAR_
           PROGRAM_NAME)
      CALL F_AUTOPAR_SET_SIZE(3,1,10)
      CALL F_AUTOPAR_SET_SIZE(2,1,10)
      CALL F_AUTOPAR_SET_SIZE(1,1,10)
      CALL F_AUTOPAR_SET_SIZE(0,1,10)
      CALL F_AUTOPAR_SET_DECL(A,4,4)
      CALL F_AUTOPAR_SET_SIZE(3,1,10)
      CALL F_AUTOPAR_SET_SIZE(2,1,10)
      CALL F_AUTOPAR_SET_SIZE(1,1,10)
      CALL F_AUTOPAR_SET_SIZE(0,1,10)
      CALL F_AUTOPAR_SET_DECL(B,4,4)
```

(下转第109页)