含数组引用的过程间数据流分析*

吉晓梅　张兆庆　乔如良　李　杰

（中国科学院计算技术研究所，北京 100080）

摘要　并行重构是发挥多处理机高计算性能的重要手段. 但是许多并行重构系统，往往不作过程间数据流分析，因而限制了含调用语句的 DO 循环的并行. 本文以中国科学院计算技术研究所并行编译组研制的 PORT(Parallelizing Optimizing Restructuring Tools)系统为背景，给出含数组引用的精确的过程间数据流分析的设计思想、关键技术及其在并行化、优化方面的应用.

关 键 词　超级编译，并行编译，并行重构，数据流分析，全局优化.

为了充分发挥超级计算机所提供的高速计算性能，利用自动并行重构技术开发超级编译器，具有重要意义. Fortran 作为一种科学计算语言，积累了大量应用程序，所以近 20 年来对并行重构的研究多数是针对 Fortran 源程序的，对串行的 Fortran 源程序做源－源变换，将其转换为带并行代码的扩展 Fortran 源程序.

本文研究工作以中国科学院计算技术研究所研制的适用于 SMS(shared－memory system)面向 Fortran 的 PORT(Parallelizing Optimizing Restructuring Tools)系统为背景.

依赖分析是并行重构的核心，而数据流分析直接影响着依赖分析的精确度，并影响着各种优化策略(例如常数传播)和并行转换的策略及效率. 由于过程间数据流分析复杂性高，以往的许多并行编译系统回避了过程间数据流分析，对程序单位的每一条调用语句，认为实参及被调用程序单位所能访问到的全局变量都因为调用语句的执行而发生了变化，这是一种最保守的假设，这种假设显然影响了数据流分析的精确度，从而降低了依赖分析的精确度，影响了优化及并行识别率，例如只可进行局部的常数传播而且不可并行含调用语句的 DO 循环.

目前，由于强调结构化的程序设计，一个模块性很强的程序往往包含大量的小程序单位，因此，过程间的数据流分析越来越显示出了它的重要性. 过程间的数据流分析要建立在调用图的基础上，同时还要考虑别名所产生的影响.

带数组引用的过程间数据流分析是 90 年代的热门话题，我们在 PORT 中实现了带数组引用的过程间数据流分析. 在进行数据流分析的过程中，精确考虑调用语句所引起的子程

* 本文 1994－05－10 收到，1994－08－15 定稿

本研究项目受国家自然科学基金资助. 作者吉晓梅，1970 年生，硕士，主要研究领域为并行编译. 张兆庆，1938 年生，研究员，主要研究领域为并行语言编译及工具环境. 乔如良，1937 年生，研究员，主要研究领域为语言编译及工具环境. 李杰，1963 年生，助研，主要研究领域为计算机软件.

本文通讯联系人：吉晓梅，北京 100080，中国科学院计算技术研究所

序对父程序的实参及公用变量的影响(此处变量是广义的,不仅包含简单变量,也包含数组变量.后面文中如不特别声明,均指广义的变量.当然,在算法处理中它们是被区分的,要作不同处理),进行全局数据流分析.因此,PORT 系统可以做全局的常数传播,可以并行含调用语句的 DO 循环.

本文基于 PORT 系统的背景,给出精确的过程间数据流分析的设计思想、关键技术及应用.

1　设计思想

Fortran 程序中程序单位之间传递数据有两种方法:一是通过哑实结合,一是通过公用变量.过程间的数据流分析也就是分析子程序的调用对父程序的影响,包括对其父程序的实参及可访问的全局变量的影响.也就是说,分析对子程序调用而言,父程序中哪些实参被引用,哪些实参被定值,哪些全局变量被引用,哪些全局变量被定值.确定被定值或引用的实参及全局变量,就可以认为若干个实参及全局变量在这个 call 语句被定值或引用,这样,call 语句就可以被认为是若干条对实参及全局变量的定值、引用语句,它可以象其他的赋值语句一样地加入过程内的数据流分析.因此,每进行一个程序单位的数据流分析,首先进行过程间的数据流分析,确定子程序调用处被定值、引用的实参或公用变量,其次,进行包含 call 语句在内的过程内数据流分析,这里的 call 语句已被认为是若干条对实参或公用变量的定值、引用语句.按上述两步所做的数据流分析是较精确的、全局的数据流分析.

由上述介绍可以看出,进行过程间的数据流分析,需要先分析子程序,再分析父程序,因此它建立在调用图的基础上,自底向上由叶子(不含任何调用的程序单位)逐层向上分析至根结点(主程序).

子程序调用对父程序的实参的影响比较容易确定,根据哑实结合的原理,与调用语句实参位置相一致的哑元若在子程序中被引用或定值,则此实参被认为在调用语句处被引用或定值.

对于公用变量,可以采用区间复盖分析法,即:将一个公用区看作一个线性存储空间,此公用区中的每一个变量按其在公用区的位置,计算出其在线性空间上所应占的位置,在各公用区的线性空间上统一分析公用量在各程序内及程序间的定值引用关系.例如:

```
real A(10),B(5,5)
integer i,j
common /x/ A,i,j,B
```

将公用区 X 视作一个线性空间,公用变量 A,i,j,B 在线性空间的位置分别为 0—39,40—43,44—47,48—147.

每分析完一个程序单位,记录此程序单位可访问的公用区的被定值及引用空间.对于一个调用语句,若程序单位中某个公用区的某个公用变量所占据的位置与被调用子程序中此公用区的某个定值或引用空间相交,则认为此公用变量在调用语句处被定值或引用.

2　关键技术

这一节首先给出过程间数据流分析所需的关键数据结构,然后介绍全局数据流分析的

流程，最后给出过程间数据流分析的关键算法.

2.1 关键数据结构 DS

DS1：程序单位的定值信息表 callinf_def，它的类型定义如下：

```
struct Callinf_def{
    int def_dump[DUMPN];              /* 程序单位中被定值的哑元位置 */
    int defnum;                       /* 程序单位中被定值的公用区数目 */
    struct DEF{
        char def_comm[VL+2];          /* 被定值的公用区名 */
        int low[MAX_COMMNAME];        /* 定值区间的下界 */
        int high[MAX_COMMNAME]        /* 定值区间的上界 */
        int deflh;                    /* 被定值区间的个数 */
        }defc[1];
    }
```

DS2：程序单位的引用信息表 callinf_use，它的类型定义如下：

```
struct Callinf_use{
    int use_dump[DUMPN];              /* 程序单位中被引用的哑元位置 */
    int usenum;                       /* 程序单位中被引用的公用区数目 */
    struct USE{
        char use_comm[VL+2];          /* 被引用的公用区名 */
        int low[MAX_COMMNAME];        /* 被引用区间的下界 */
        int high[MAX_COMMNAME]        /* 被引用区间的上界 */
        int uselh;                    /* 被引用区间的个数 */
        }usec[1];
    }
```

2.2 数据流分析的流程

数据流分析的基本流程图如下：

```
while (有未处理的程序单位) {
    1. 初始化;
    2. 子程序对父程序的信息传递;
    3. 求标量 ud、du 链及数组 du 链;
    4. 求标量易名链;
    5. 寻找归纳变量;
    6. 求此程序的定值、引用信息;
    }
```

在进行数据流分析过程中，首先分析调用语句所导致的过程间数据流传播(步骤2)，将调用语句视为系列实参及公用变量的定值、引用语句，然后进行包括 call 语句(此时的 call 语句实质上已转化为系列定值、引用语句)在内的过程内数据流分析(步骤3—5)，也就是沿着控制流图传播数据流信息，每分析完一个程序单位后，求出其定值、引用的哑元及公用变量信息(记入 callinf_def 及 callinf_use 结构)，以便于在下一个循环阶段，向其父程序传递信息(步骤6). 这个流程是一个循环过程，每次循环处理一个程序单位，循环次序是按照调用图自底向上进行的.

上述流程图中，步骤2、6是过程间数据流分析的关键两步，下面我们将介绍其主要算法.

2.3 过程间数据流分析的关键算法

过程间数据流分析已被人们广泛讨论，参考文献[1—4]给出了求过程间数据流分析的算法，但多数算法是不敏感于控制流图的，因此是不精确的. 在这里，我们给出求解过程间数据流分析的敏感于流图的精确算法.

算法1. 求程序单位 P 的定值、引用信息

```
dumpn=0;          dumpuse=0;
for every 哑元 V{
   if V 在程序单位 P 被定值
     callinf_def[p]→def_dump[dumpn++]=V 在形参表中的位置;
   if V 在程序单位 P 中被引用
     callinf_use[p]→use_dump[dumpuse++]=V 在形参表中的位置;
   }
callinf_def[p]→def_dump[dumpn]=0;
callinf_use[p]→use_dump[dumpuse]=0;          /* 以'0'表示结束 */
for every 公用变量 V{
   if V 在程序单位 P 中被定值{
     将 V 所在公用区名填入 callinf_def 中 defc 结构的 def_comm 域;
     将 V 在公用区中占据的起止位置填入 callinf_def 中 defc 结构的 low,high 域;
     }
   if V 在程序单位中被引用{
     将 V 所在公用区名填入 callinf_usc 中 usec 结构的 use_comm 域;
     将 V 在公用区中占据的起止位置填入 callinf_use 中 usec 结构的 low,high 域;
     }
   }
      将 P 中所调用的子程序的 callinf_def 及 callinf_use 结构中有关公用变量信息合并入 callinf_def[p]及 call-
   inf_use[p];
```

注意:上述算法的最后一步是鉴于下述原因:一个公用变量 V 可能会通过一个 V 在其中不可见的程序单位 P 而进行传递.

例如:

```
subroutine        test
common i,j
…
call A(x)
…=j
…
end
subroutine A(F)
common i
…
call B(F)
…
end
subroutine B(F)
common i,j
…
j=…
j=…
…
end
```

公用变量 j 在程序单位 A 中不可见,但 j 在程序单位 B 中的定值通过程序单位 A 而传递给程序 test. 因此,我们在收集程序单位 A 的定值,引用信息时,也要合并其子程序 B 中有关公用变量的定值、引用信息,只有这样,才能将公用变量 j 的信息成功地传递给 test 程序单位.

算法2. 子程序 P 对父程序 q 的信息传递

```
dc=0;
while (num=callinf_def[p]→def_dump[dc++])
     找到第 num 个实参,将其置为被定义;
```

```
uc=0;
while (num=callinf_use[p]→use_dump[uc++]){
      找到第 num 个实参,将其置为被引用,
      若此实参为表达式,则置每个操作数被引用;
      }
for (comn=0;comn<callinf_def[p]→defnum;comn++){
   strcpy(comname,callinf_def[p]→defc[comn].def_comm);
   求出父程序 q 中 comname 公用区的所有公用变量 V;
   for every 公用变量 V{
      求出其在公用区这个线性空间所占据的起止位置 offset 及 endoffset;
      if (区间(offset,endoffset)与程序单位 p 中 comname 公用区的某个定值区间相交)
        置公用变量 V 被定值;
        }
      }
for (comn=0; comn<callinf_use[p]→usenum;comn++){
   strcpy(comname,callinf_use[p]→usec[comn].use_comm);
   求出父程序 q 中 comname 公用区的所有公用变量 V;
   for every 公用变量 V{
      求出其在公用区这个线性空间所占据的起止位置 offset 及 endoffset;
      if (区间(offset,endoffset)与程序单位 p 中 comname 公用区的某个引用区间相交)
        置公用变量 V 被引用;
        }
      }
```

3　过程间数据流分析在优化及并行化中的应用

经过过程间的数据流分析后,可以支持全局常数传播和含调用语句的 DO 循环并行化,下面我们举两个例子.

例1:

```
          PROGRAM TEST
          DATA C4/2.0/
          CMC=1.0E+1
          RR2=1.4
          LOOP=10
          C1=C4
          YY=CMC*(RR2+LOOP)
          CALL PROT (LOOP)
(S1)      CALL BUSH (LOOP)
          N2=10
(S2)      CALL BUSH (LOOP+N2)
          CALL
          END

          SUBROUTINE PROT(MM)
          INTEGER F12,F15
(S3)      F12=MM+10
          CALL CLINTON (F12)
(S4)      F15=F12*4
          PRINT *,F15
          END

          SUBROUTINE CLINTON(XX)
          INTEGER XX,IA(100)
          DATA IA/100*5/
(S5)      LL=XX
```

```
        L1=10
        L3=1
        DO 1 I=1,LL+L1
(S6)    L3=L3+XX
(S7)    IA(I)=IA(XX)*L1
  1     CONTINUE
        L4=L3*5
        PRINT *,IA,L4
        END

        SUBROUTINE BUSH(YY)
        INTEGER YY
        DIMENSION IA(100)
        DATA IA/100*2/
        L1=10
        DO 1 I=YY,3*YY-10
        IA(I)=IA(I+4)*L
  1     CONTINUE
        KK=YY+100
        PRINT *,IA
        END
```

通过过程间数据流分析,可以得出 PROT、BUSH 及 CLINTON 子程序都不修改哑元的值,因此可越过对这些程序单位的调用语句而进行常数传播(例 S1,S2,S4语句)同时也可进行与哑实结合相关的过程间的常数传播(例 S3,S5,S6,S7语句).

经全局常数传播后的代码如下:

```
        PROGRAM TEST
        TEST C4/2.0/
        CMC=1.DE+1
        RR2=1.4
        LOOP=10
        C1=2.0
        YY=1.14E+02
        CALL PROT(10)
        CALL BUSH(10)
        N2=10
        CALL BUSH(20)
        END

        SUBROUTINE PROT(MM)
        INTEGER F12,F15
        F12=20
        CALL CLINTON(20)
        F15=80
        PRINT *,80
        END

        SUBROUTINE CLINTON(XX)
        INTEGER XX,IA(100)
        DATA IA/100*5/
        LL=20
        L1=10
        L3=1
        DO 1 I=1,30
        L3=20*I+1
```

```
      IA(I)=IA(20)*10
1     CONTINUE
      L4=L3*5
      PRINT *,IA,L4
      END

      SUBROUTINE BUSH(YY)
      INTEGER YY
      DIMENSION IA(100)
      DATA IA/100*2/
      L1=10
      DO 1 I=YY,YY*3-10
      IA(I)=IA(I+4)*10
1     CONTINUE
      KK=YY+100
      PRINT *,IA
      END
```

如果仅做过程内的常数传播，则 S1,S2,S3,S4,S5,S6,S7语句均不能进行常数传播.

例2：

```
      PROGRAM TEST
      REAL A(10,10),B(10,10)
      READ A,B
      DO I=1,10
      CALL SUM(A,I)
      CALL MUL(A,B,I)
      ENDDO
      END

      SUBROUTINE SUM(X,I)
      REAL X(10,10)
      S=0.0
      DO J=1,10
          S=S+X(I,J)
      ENDDO
      END

      SUBROUTINE MUL(X,Y,I)
      REAL X(10,10),Y(10,10)
      S=0.0
      DO J=1,10
      S=S+X(I,J)*Y(I,J)
      ENDDO
      END
```

经全局流分析可知，子程序 sum,mul 对于哑元都仅做了引用而未定值，因此主程序中 DO 体内的两条调用语句之间以及与父程序之间均不存在依赖关系，所以 DO 循环可以并行. 经并行重构后主程序的并行代码为：

```
      PROGRAM TEST
      REAL A(10,10),B(10,10)
      READ A,B
C$DOACROSS SHARE(A,B),LASTLOCAL(I),MP_SCHEDTYPE=SIMPLE
      DO 120 I=1,10
      CALL SUM(A,I)
      CALL MUL(A,B,I)
120   CONTINUE
```

```
END
```

子程序 sum 及 mul 均不可并行,并行重构后代码不变.

PORT 系统对计算中心及国家地震局所提供的 benchmark 有良好的并行识别率,而且它对于国际通用的 perfect 及 livermore 测试程序也达到了较好的加速比,可与国际知名的并行重构系统 PFA 相媲美. 精确的全局流分析即是 PORT 系统的关键技术之一.

4 结　论

PORT 系统实现了较精确的含数组引用的全局数据流分析,它不仅进行过程内的数据流分析,而且进行过程间的数据流分析. 通过全局数据流分析,可以进行全局常数传播,提高依赖测试精确度;可以并行含调用语句的 DO 循环;并可望用于分布式系统的子程序一级并行化. 数据流分析还采用了一些优化算法,降低时空开销,以达到实用性.

参考文献

1 Barth J M. A practical interprocedual data flow analysis algorithm. Comm. Assbc. Computing Machinery, Sept. 1978, **21**(**9**): 724—726.

2 Cooper K D, Kennedy K. Efficient computation of flow—insensitive interprocedural summary information——a correction. SIGPLAN Notices, April 1988, **23**(**4**): 35—42.

3 Hu Shibin. Comment on cooper and Kennedy's flow—insensitive interprocedural summary information computation algorithm. ACM SIGPLAN NOTICES, May 1993, **28**(**5**): 3—8.

4 Zima H P. Supercompilers for parallel and vector computers. New York: ACM Press, 1990.

THE INTERPROCEDUAL DATA—FLOW ANALYSIS WITH ARRAY REFERENCE

Ji Xiaomei　Zhang Zhaoqing　Qiao Ruliang　Li Jie

(*Institute of Computing Technology, The Chinese Academy of Sciences, Beijing* 100080)

Abstract　Parallelizing restructuring has become an essential means to exploit the vast computing power of modern supercomputers. But many parallelizing restructuring systems ignore or simplify the interprocedural data—flow analysis, so they can not transform a do—loop which contains call statements into parallel code. This thesis introduces a practical interprocedual data—flow analysis in the front—end of the parallelizing optimizing restructuring tools (PORT), which was developed at the Institute of Computing Technology, The Chinese Academy of Sciences. It concentrates on the interprocedual data—flow analysis's idea, key techniques and application.

Key words　Supercompiling, parallelizing compiling, parallelizing restructuring, data—flow analysis, global optimizing.