一个过程间数据流分析的框架*

郁卫江 朱根江 谢 立

(南京大学计算机科学系 南京 210093)

(南京大学计算机软件新技术国家重点实验室 南京 210093)

摘要 本文提出一个过程间数据流分析的框架. 它将层次式任务图 HTG(hierarchical-task graph)用于程序功能并行性的表示与挖掘. 在框架中定义了过程表 ProcTable 和二叉树形式的过程调用图 BCG(binary_call_graph),以使算法的时空代价最小.

关键词 并行编译,过程间数据流分析,过程调用,层次任务图,数据流分析,数据依赖分析.

中图法分类号 TP311.11

大规模并行处理系统能有效地利用资源,使其获得比单独计算机更强的解决高度复杂的计算和处理问题的能力. 在目前及可预见的未来是实现高性能计算的主要途径. 但是,并行计算程序编制困难,而且现有的大规模并行机的应用针对性很强,受到软件限制,使用也不方便,其峰值速度和实际效率有很大差距. 并行编译是解决上述问题的有效途径,其主要工作是挖掘顺序程序的并行性,使之并行化.

一般来讲,发掘一个程序并行性主要有两个方面. 一个是数据并行,即并行性是针对大量数据上的迭代操作而言. 另一个是功能并行,它是非循环一级的并行,主要是指并发地执行一些不同的子程序调用或程序段. 数据并行一直是自动并行编译的重点,尤其是针对循环一级,其相应的依赖分析技术已经基本成熟. 功能并行性是近期国际上并行编译研究的新热点. 功能并行性的挖掘是建立在对任务数据流及控制流分析的基础之上,并行单位可被看作是一个任务. 在实际的程序中,一个过程调用可视为一个任务,所以,过程间数据流分析 IPA (inter-procedural analysis)对挖掘程序功能并行性是非常重要的.

过程间数据流分析是必要的. 若不然,在并行化过程中对程序进行重构时,编译器没有关于被调用过程的信息,就不得不对被调用过程作一些保守的假设,即在该调用点可见的所有变量都可能被使用和改变,这就抑制了一些可能的并行. 在使用大量公共变量或传址参数的程序中,分析出这种假设和实际情况之间的差别就很重要了. 所以过程调用引起的副作用的详细信息成为许多研究并行编译的人关注的问题. 特别是在每个过程调用点,由于过程的

---

* 本文研究得到国家863高科技项目基金资助. 作者郁卫江,1973年生,硕士生,主要研究领域为并行编译. 朱根江,1960年生,博士,副教授,主要研究领域为并行编译. 谢立,1942年生,教授,博士导师,主要研究领域为分布与并行计算机系统.

本文通讯联系人:郁卫江,南京 210093,南京大学计算机科学系

本文1996-11-08收到修改稿

执行,到底哪些变量被使用,哪些又被改变,这些信息搜集的越详尽,数据依赖分析的结果就越精确.就面向对象程序设计语言(如C++)而言,其数据与方法的封装特征及其基于对象间发送消息的编程模型,使研究过程调用引起的数据依赖问题显得尤为重要.IPA技术正是用于突破"过程"这一界限,使程序员使用过程编程和编译器进行优化编译达到一种平衡.

10多年来,研究人员提出了许多IPA方法.内联(In-line)扩展方法是IPA的一种最简单方法,即把过程体替换到每个调用点.它使程序膨胀到几乎不可接受的程度,实际上很少单独采用.Interval分析则是在对程序进行Interval图构造的基础上,利用"转换函数"分析输入参数以计算过程调用产生的影响.此方法在FORTRAN程序中用的较多.[1]常数传递(ICP)方法是通过把变量的常数值穿过调用边界传递到过程内,使在程序执行时以常量形式出现,从而利于并行化.[1~3]区域概要信息组(Region Summary Group)方法[4]通过分析循环内过程调用的数组信息,实现跨迭代的数据依赖分析.Cooper和Kennedy[5]提出一种称为Binding-Multigraph的方法来解决IPA问题.它用图形表示,将相关的形一实参数联系起来,达到参数传递的目的.Mohd-Saman等人采用基于Bernstein集的方法,得到了较为精确的过程调用数据流信息.[6]指针分析一直是并行编译中的难点.Wilson等人[7]是通过一种部分转换函数的方法力图缩小指针引用的变量集,以期得到更大的数据流依赖信息.

本文提出了一种过程间数据流分析IPA的框架.我们将层次式任务图HTG用于程序功能并行性的表示与挖掘.同时定义了过程表ProcTable和二叉树形式的过程调用图BCG,用于搜集过程调用的信息和在被调过程体内变量的使用对该调用点的数据集产生的影响.

对过程进行数据流分析必然涉及别名问题.当两个变量同时引用相同或重叠的存储区域时,它们就互为别名.C/C++语言的指针作为参数传递会引起别名问题.我们发现在实际的C/C++程序中别名问题较少,而主要是指针问题,这与别名问题不尽相同.我们采用位向量表示变量,而不是所谓的"存储区域".在过程体内先归总数据信息,然后把参数替代进来,再合并数据信息,从而解决了别名问题.

1 层次任务图HTG和过程的数据表示

1.1 HTG图

文献[6]提出的BSs-IPA方法没有给出源程序的中间表示,而实际上挖掘程序的功能并行性必须要有好的中间表示.我们采用了改进的层次式任务图HTG(hierarchical-task graph)表示技术.[8]

编译器详细分析源程序得到控制流图CFG(control-flow graph).[1]HTG是基于CFG上构造的,其结点可分为简单结点、复合结点、循环结点、过程调用结点、系统库函数结点(目前暂不考虑).

HTG是一种功能很强的并行化程序中间表示,它包含了不同类型和层次的并行性,可以用来生成和优化并行资源和并行机器代码.它使动态调整并行粒度成为可能.

数据流分析包含对各种HTG结点数据使用和修改情况的搜集工作,而IPA的工作就是把HTG图中的Call结点的数据信息(如读写变量集、实参、调用对象等)收集起来,分析该Call结点进一步扩展而成的HTG图的所有数据流信息,然后汇总并返回到调用点,得到

互不相交的只读、只写、读且写变量集,供后阶段依赖分析使用.

需要说明的是,数据流分析基于一个结构化的流图,适用于可归约程序.[1]一般的结构化程序都是可归约的.不可归约程序可以通过结点分裂等方法解决.由于一个 HTG 可以是一个很复杂的图,在进行过程间数据流分析时,就产生了一个结点遍历问题.我们采用了深度优先遍历树的方法来处理流图中的每个结点的数据,在时空代价上要优于其它遍历流图的方法.[9]

1.2　数据表示

IPA 中数据表示是非常重要的.首先,我们把基本块 B(一组顺序语句)的数据表示成如下两种集合:$MOD(B)$和$USE(B)$.对任意一个基本块 B 及一个变量 v:

$v \in MOD(B) \longleftrightarrow$执行 B 改变 v 的值

$v \in USE(B) \longleftrightarrow$执行 B 使用了 v 的值

显然,在构造 CFG 时我们是不难得到每个基本块的 MOD 与 USE 集合的.一般的方法是将过程间数据流也分为 MOD 与 USE 集[5],但是为了更精确的表示数据信息,我们采用文献[6]中提到的基于 Bernstain 集的方法,并加以改变.它是在上述数据集合的基础上得到的,并且可以扩充到对数组下标变量的表示,适合于我们的 IPA 分析.

一个基本块 B 包含以下集合(Si 为 B 所包含的一条语句):

- RD 集——执行 B 时只读变量集:　$RD(B)=\bigcup_{Si\in B} USE(Si)-\bigcup MOD(Si)$
- WT 集——执行 B 时只写变量集:　$WT(B)=\bigcup_{Si\in B} MOD(Si)-\bigcup_{Si\in B} USE(Si)$
- RW 集——执行 B 时读且写变量集:$RW(B)=\bigcup_{Si\in B} USE(Si)\cap\bigcup_{Si\in B} MOD(Si)$

我们感兴趣的是 Flow-sensitive 分析,即过程体内所有路径的执行都一定引起哪些数据的读或写;与此相应的是 Flow-insensitive 分析,即一个过程调用可能引起的数据读写.[5,6,9]进行 Flow-insensitive 分析可使算法简洁明了,而有关控制流的分析我们另外处理.两者的结合便可得到运行时刻的执行条件,计算出更为精确的数据依赖关系.

这样,对一个过程在 S 点调用,我们有以下一组公式:

$$Call\_RD(S)=Local\_RD(S)\cup[\bigcup_{e=(p,q)\in S} map1_e(G\_RD(q))] \tag{1}$$

$$Call\_WT(S)=Local\_WT(S)\cup[\bigcup_{e=(p,q)\in S} map2_e(G\_WT(q))] \tag{2}$$

$$Call\_RW(S)=Local\_RW(S)\cup[\bigcup_{e=(p,q)\in S} map3_e(G\_RW(q))] \tag{3}$$

$Call\_XX$(XX 可以是 RD,WT,RW 之一,下同)是过程 p 在调用点 S 处的读/写/读且写变量集.$Local\_XX$ 则是 S 所在语句的相应变量集,一般是实参变量(XX 为 RD 时)或过程作为函数使用的一个赋值语句所涉及的其它变量.函数在调用点 $e=(p,q)$(q 是 p 内的任一过程调用)把 q 的名字域和参数映射成 p 的相应变量.可见,$Call_XX(S)$包含了 S 的局部信息加上 S 中任一过程调用执行的结果.实际上,(1)~(3)得到的数据集仍需进一步计算其交集,以得到互不相交的 RD,WT,RW 集合.这一点对后面讨论的数据依赖分析是重要的.

考察(1)~(3)式,我们关心的是 $G_XX(p)$该如何得到.先给出下面的公式:

$$G\_RD(p)=D\_RD(p)\cup[\bigcup_{e=(p,q)} map1_e(R\_RD(q))]\cup[\bigcup_{e=(p,q)}(G\_RD(q)\cap\overline{LOCAL(q)})] \tag{4}$$

将上式的 RD 依次换成 WT,RW,$map1$ 换成相应的 map 函数,可得相应的数据流方程.(4)式表明过程 p 的一般 RD 集($G\_RD(p)$)等于 p 内直接使用的变量集($D\_RD(p)$)加上 p 内任一过程调用传递参数所使用的变量集($map1(R\_RD(q))$,$map1$ 被简化为形-实参数对),加上该内部过程的一般 RD 集(递归思想),并去掉 q 的局部变量.这样,一个大问题被分解成若干小问题,逐一求解.

WT,RW 的计算是类似的,不同之处在于 map 函数的构造要考虑传址/引用形参和别名问题,后文将会讨论.

要注意每次计算出 q 的 RD,WT,RW 集之后,应对 q 的调用点 S 重新计算上述 3 种集合,使其互不相交.最终,我们得到了 HTG 任一层次的数据流信息,这是表示功能性并行及控制并行粒度所要求的.

2 过程表示的框架

2.1 过程表 ProcTable

HTG 中的每个结点数据结构包含一个结点类型域,当为 *Call* 结点时,则有一个相应的实参链及指向过程 HTG 结构的指针.

把实参放在调用结点处,同时建立一个过程表 ProcTable.这样做有几个好处:①分析源程序方便,避免了在分析过程中,每次调用都把整个程序调入内存,只需要过程的最主要信息;②使一个静态过程表表项可以动态地与多次调用相"结合",从而减少了数据复制;③利于解决别名问题.

ProcTable 包含了过程的确定信息,如过程名、形参、在其被调用过程中的调用点、类表下标、HTG 结点个数等等.我们的过程表支持"Flow-sensitive"分析,即与控制流结合起来,这是通过过程调用图的结点指针来实现的.

内联扩展方法是 IPA 的一种最简单方法,即把过程体替换到每个调用点.此方法虽有利于较细粒度的并行化且消除了过程调用开销,但它使程序膨胀到几乎不可接受的程度,实际上很少单独采用.BSs_IPA 方法完全不采用.但是,我们发现实际的程序有不少很短小的过程.当 HTG 结点个数小于 5 时,我们使用该方法,实验表明这样做是合理的.

2.2 过程调用图 BCG

过程调用图中的结点表示程序中被调用的过程,结点的连线表示调用关系.一个 BCG (binary_call_graph)实际上是以 *main*() 作为根结点的一棵调用树,反映出整个程序执行的过程调用状况,其中树叶结点为无其它过程调用的过程或递归调用的过程.

结点中用适当的域表示数据信息,在后面的自底向上分析中,收集的数据信息填充在结点的相应域里.实现时,我们采用二叉树的方法表示过程调用图.这样做使得调用图的遍历算法较为简单.对于每个结点 C,其左孩子表示在 C 内调用的第 1 个过程,若空,则意味 C 本身不再包含任何过程调用,其右孩子表示 C 调用点所在过程的下一个邻接过程调用,若空,则意味 C 是所在过程的最后一个过程调用.图 1 给出一个例程的 BCG.

递归过程这样处理:设一个递归过程调用链为 $P\rightarrow\ldots\rightarrow X'\rightarrow\ldots\rightarrow X$,则该链的最后一个过程调用 X 必定已经在调用链中(设为 X'),且 X 与 X' 的参数相同.分析时,只要检查出某一过程调用 Q 已出现在调用链中,且有相同的实参,就表明到达了递归调用链的末尾,

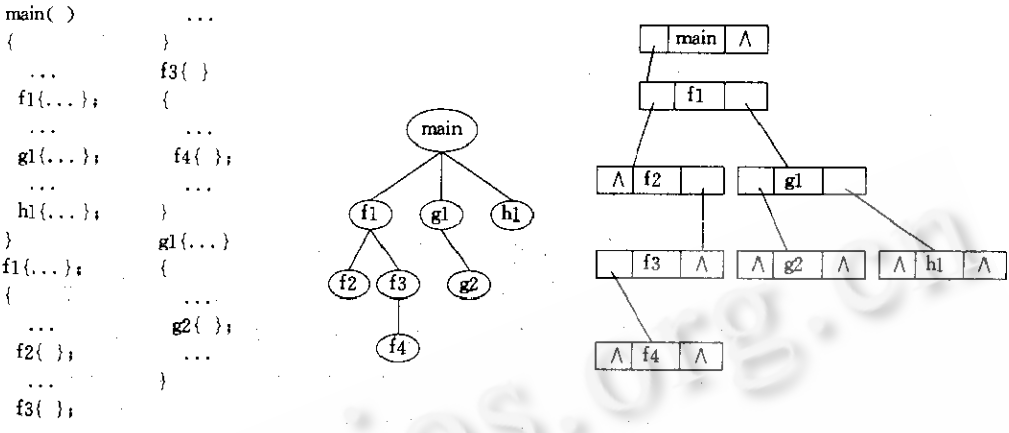

图1 Binary_Call_Graph

并且 Q 就是 BCG 的树叶.

HTG 图、ProcTable 及 BCG 三者构成了 IPA 的框架. 图 2 是一个过程调用的框架.

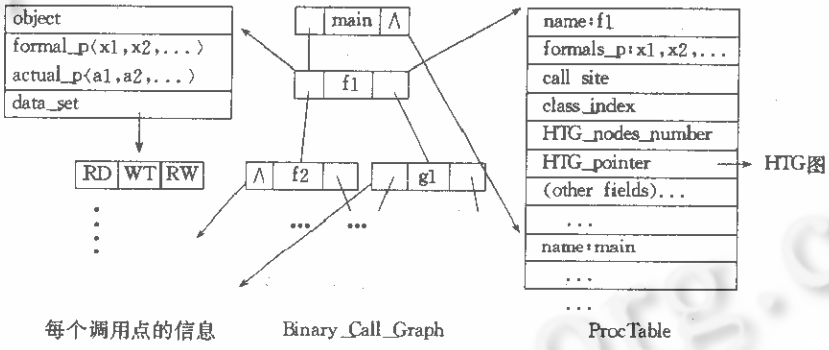

图2 过程调用的框架

3 过程间数据流分析算法 HTG_IPA

我们实现了一个过程间数据流分析的算法,称作 HTG_IPA,其基本思想是在一个过程内的任何过程调用点(包括实例对象的方法的使用)处,分析被调用过程体,然后把搜集到的信息返回到调用点. 从调用树的角度看就是把信息(实参、对象、作用域等)从树根传递到树叶,然后把搜集整理后的信息自树叶向上传递.

HTG_IPA 算法.

算法 1. 过程间数据流分析 HTG_IPA

```
1. init_ProcTable();
2. /* create Call Graph */
3. {
4.     fill_item_fields();
5.     transmit(class_index,object);
6.     for each call site
7.         create(actual_parameter,formal_parameter);
8.     create_call_graph( ) /* recursively */;
```

```
9. }
10. trans_class( object,HTG_head );
11. form_new_AFs( );
12. trans_actual( actual_parameter ); /* recursively */
13. preorder_travel( Binary_Call_Graph );
14. {
15.         for each node Ni
16.                 merge( Ni );
17. }
```

第1行是初始化ProcTable,计算每个过程表项的每个HTG结点的RD,WT,RW集,即静态收集信息.第2行到第9行是创建Binary_Call_Graph,其中第5行是把每个过程调用的对象信息(记录在HTG的*Call*结点的相应域内)传递给该调用图中的结点.为了讨论方便,前面我们一直没有涉及C++语言的特殊之处.实际上我们分析了对象数据信息的传递问题.建立了一个类表,在ProcTable中有该过程所属类的类表下标.在Call graph中有调用该过程(方法)的对象.全局过程的对象规定为第"0"类对象,即把它们与类的方法等同对待,以便于算法的通用与简洁.变量用位向量来表示.用若干位表示该变量是哪一个类的成员或是某一个方法的形参或局部参数.第10～12行实参、对象信息传递到被调用过程体内,第12行表明这是个递归的处理过程.第2～12行是把数据信息自顶向下传,第13～16行则是把数据信息自底向上传.过程*Merge*()把每个过程的信息归并整理,得到在调用点处应看到的信息.

算法2. 过程信息的归并整理*Merge*()

```
Merge(p):
输入:过程调用图的一个结点所对应的过程 p
输出:p 的 RD,WT,RW 集合
    {
    get_all(p,new_data);
            /* 分别汇总该过程体内所有 HTG 结点的 RD,WT,RW 集合; */
    del_local( new_data);
            /* 删除局部变量 */
    summary( D_RD(p),new_data);
            /* 将调用点数据信息与刚得到数据集合归并; */
    compute(new_data);
    compare(new_data,new_data');
            /* 比较 new_data 的 RD,WT,RW,得到最终的数据集;
                    比较时,①若 a∈RW,则 a∉RD,a∉WT;
                            ②若 a∈RD,且 a∈WT,则把 a 从 RD,WT 中删去,再把 a 加入到 RW 中去.
            */
    }
```

在归并过程中,仍需保留每个HTG结点的新的读写集,这是因为HTG是层次式的,要求每一层上都有数据信息.

4 数据依赖分析

两个HTG结点x,y,若x和y对同一内存地址访问,且至少有一个为写操作,则说x和y数据依赖.在HTG图中有3种情况:(1) 从x可达y;(2) 从y可达x;(3) x,y之间无路径.

前两种情况下若x和y数据依赖,则分别记为$x\delta dy$和$y\delta dx$,后一种情况则说明x和y

的数据依赖与否无关紧要,我们不考虑这一情况.这样,我们得到了每个 HTG 图对应的数据依赖图 $DDG=(Vd,Ed)$,其中 Vd 即为 HTG 的结点,$(x,y)\in Ed$ 当 $x\delta dy$.

在已得到的较为精确的过程间数据流信息的基础上,我们求出每一个 HTG 每个结点 n 的可达结点集 $Reach(n)$,然后分析两个结点的数据依赖关系.

对于任两个结点 i,j,其中 $j\in Reach(i)$,i 和 j 的 RD,WT,RW 集已求出,当满足下列条件时:

$$(WTi\cup RWi)\cap(RDj\cup RWj)=\varnothing$$
$$(RDi\cup RWi)\cap(WTj\cup RWj)=\varnothing$$
$$(WTi\cup RWi)\cap(WTj\cup RWj)=\varnothing$$

则 i,j 无数据依赖,否则 i 和 j 数据依赖,记为 $id\delta j$.

求解上述测试条件,我们可以得到任一过程调用点与同一层次上其它 HTG 结点的数据依赖性.当然,由于执行条件的变化,过程调用分析得到的数据流信息是实际读写集的超集,并且要和控制流信息结合起来,才能得到更为精确的依赖关系.有关研究,另文讨论.

5　实例和性能分析

5.1　实现概述

在 SUN SPARC-2 工作站(操作系统为 SunOS)和 WYSE 7 000i 并行机(操作系统为 SCO UNIX)上,我们实现了一个基于 C++的自动并行编译工具——NUAPC.[10~12]我们使用 LEX 和 YACC 对 C++源程序进行词法和语法分析,得到其中间表示 HTG,经过控制流和数据流分析,得到源程序的控制依赖图 CDG 和数据依赖图 DDG.本文提出的过程间数据流分析框架用于 DDG 的生成过程.

5.2　实　例

本节给出一段程序(如图 3 所示),应用前面所述的 IPA 框架及 HTG_IPA 算法分析其过程间数据流的情况.

```
class A{
variable count,data;
    ...
    A();
    Input(variable in);
    Processdata();
    Get();
    Output();
    ...
}
```

```
A::Input(variable in){
      S7:count=in;
    }
    A::Processdata(){
    variable i;
      if(count>0)
      for(i=1;i++;i<count)
        data+=i*i;
      S8:Output();
    }
    A::Get(){
      return this→data;
    }
    A::Output(){
      printf("%d",data);
    }
```

```
Global A al;
...{
      variable p,q,i;
        ...
      S1:f(p,q);
      S2:i=p+q;
        ...
}
f(variable x,& y){
    variable i=10;
      S3:x=x+i*y;
      S4:al.Input(x);
      S5:al.Processdata();
      S6:y = al.Get();
    }
```

图 3

首先创建 BCG, ProcTable 及每个调用点的初始信息,如图 4 所示.

建立每个调用点的形-实参数对,如 $S1:(p,x)(q,y)$, $S4:(x,\text{in})$. 各 Si 的数据集如下:

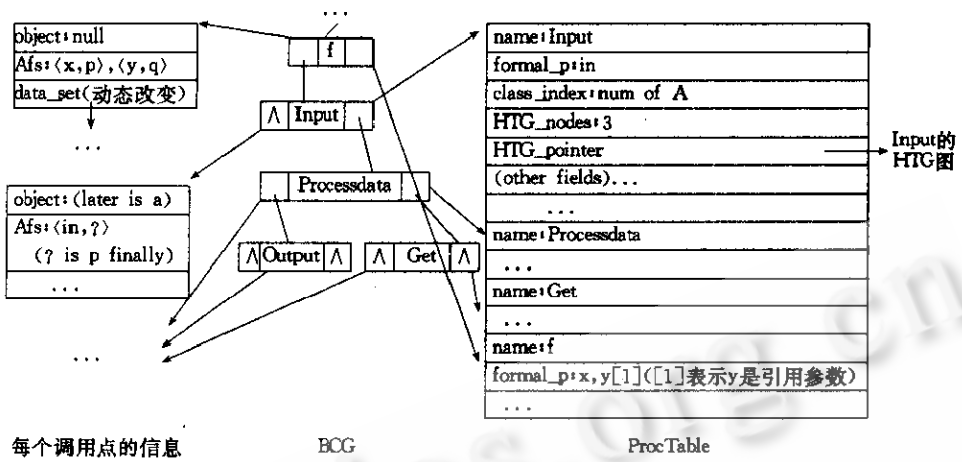

图4 一个实例的IPA框架

	S1	S2	S3	S4	S5	S6	S7	S8
RD	p,q	p,q	f.i,f.y	f.x	(a1)	(a1)	input.x	—
WT	—	i	—	—	—	f.y	count	—
RW	—	—	f.x	—	—	—	—	—

从BCG的 f 开始,将实参及对象信息传递下去.我们得到各 Si 的数据集如下:

	S1	S2	S3	S4	S5	S6	S7	S8
RD	p,q	p,q	f.i,q	x/p	(a1)	(a1)	in/p	(a1)
WT	—	i	—	—	—	q	a1.count	—
RW	—	—	x/p	—	—	—	—	—

最后从下(树叶)往上(树根)遍历二叉树,对每个访问接点执行 *Merge*()算法.各个 Si 的最终数据集如下:

	S8	S7	S6	S5	S4	S3	S2	S1
RD	a1.data	a1.Input.in/p	a1.data	a1.count	f.x/p	f.i,q	p,q	p
WT	—	a1.count	q	—	a1.count	—	i	—
RW	—	—	—	a1.data	—	f.x/p	—	a1.count,a1.data,q

a/b 表示动态分析时为 b 变量,而当HTG扩展到该层时为 a 变量,其值为 b.

5.3 相关工作比较

Mohd-Saman 等人提出了一种基于 Bernstain 集(BSs)的 IPA 通用方法[6],其目标是把信息从过程调用点传递到过程体内,并把新的信息返回到调用点,以期得到较为详尽的过程间数据流信息.文献[5]中给出了一般过程式语言的过程调用点数据流方程,并提出了一种称为 binding multi-graph 的数据结构来表示形参与(多次)实参的匹配,对该图进行深度优先搜索,可获得线性的运行时间,我们对这两个方法进行了改进:

(1)BSs-IPA 方法没有一个好的流图表示,我们引入层次任务图 HTG 来表示和发掘程序的功能并行性.(2)我们定义了一种过程表结构 ProcTable,精简地表示出 IPA 所需的过程信息,很好地表示了形-实参数,特别是针对过程参数被进一步作为参数"纵深"传递下去

的情形. 实验表明,ProcTable 有助于 IPA 算法的简化. (3)我们采用二叉树表示过程调用图(Call Graph),在时空开销上优于 BSs-IPA 方法. (4)我们并行编译的源语言是 C++语言,对象信息在过程间的传递与返回是 IPA 的重要内容,通过 ProcTable 的合适表示及相应的算法来解决这个问题. 另外,内联扩展是 IPA 的一种最简单的方法,在适当的情形下可以采用.

6 结　论

本文提出了一种过程间数据流分析的框架,目标是挖掘程序的功能并行性. 我们使用层次式任务图 HTG,它适合于功能性并行的表示. 为了使算法的时空代价最小,我们定义了过程表 ProcTable 和二叉树形式的过程调用图 BCG,同时采用改进的基于 Bernstein 集的数据表示方法,使数据的表示简洁明了,并为下一阶段的数据依赖分析打下了基础. 我们还对 C++这样的面向对象语言的过程间数据流分析进行了研究.

虚函数及指针是数据分析的难点. 文献[7]提出了一些方法及改进. 但在实际程序中,由于指针的存在常常分析出数据冲突的结果,大大降低了并行的可能.

今后的工作是改进和扩充我们的算法,特别是加入对数组参数的分析.

参考文献

1 Aho A V, Sethi R, Ullman J D. Compilers principles, techniques and tools. Addison-Wesley, 1986.

2 Binkely David. Interprocedural constant propagation using dependence graphs and a data-flow model. In: International Conference on Compiler Construction, 1994. 231～242.

3 Carini Paul R, Hind Michael. Flow-sensitive interprocedural constant propagation. ACM Sigplan Notices, June 1995. 23～31.

4 Iitsuka Takayashi. Flow-sensitive interprocedural analysis method for parallelization. In: Architecture and Compilation Techniques for Fine and Medium Grain Parallelism, 1993.

5 Cooper K D, Kennedy K. Interprocedural side-effect analysis in linear time. In: Proceedings of the ACM Sigplan'88 Conference on Program Language Design and Implementation, Atlanta, GA, June 1988. 57～66.

6 Mohd-Sanam M Y, Evans D J. Interprocedural analysis for parallel computing. Parallel Computing, 1995, **21**: 315～338.

7 Wilson R P, Lam M S. Efficient context_sensitive pointer analysis for C programs. ACM Sigplan Notices, June 1995. 15～27.

8 Girkar M B. Functional parallelism theoretical foundations and implementations[Ph. D. dissertation]. Center for Supercomputing Research and Development, University of Illinois, 1991.

9 Callahan David. The program summary graph and flow-sensitive interprocedural data flow analysis. In: Proceedings of the ACM Sigplan'88 Conference on Program Language Design and Implementation, Atlanta, GA, June 1988.

10 Zhu Genjiang, Xie Li, Sun Zhongxiu. PRG: procedure reference analysis in extracting non-data parallelism. In: 1996 IEEE Second International Conference on Algorithms & Architectures for Parallel Processing, Singapore, June 1996. 76～83.

11 Zhu Genjiang, Xie Li, Sun Zhongxiu. A path-based method of parallelizing C++ programs. ACM Sigplan Notices, 1994, **29(2)**: 54～64.

12 朱根江,谢立,孙钟秀. 一种基于非正规域的区域依赖关系分析法. 计算机学报,1994, **17(3)**: 168～175.

A FRAMEWORK FOR INTERPROCEDURAL DATA-FLOW ANALYSIS

YU Weijiang ZHU Genjiang XIE Li

(*Department of Computer Science Nanjing University Nanjing* 210093)
(*State Key Laboratory of Novel Software Technology Nanjing University Nanjing* 210093)

Abstract IPA (interprocedural analysis) is the key for parallelization of the serial programs. This paper discusses a framework for IPA. A HTG (hierarchical-task graph) is used to find the functional parallelism. Based on two new data structures, called ProcTable and Binary_Call_Graph, the IPA algorithm reduces cost both in time and space compared with other techniques.

Key words Parallelizing compilers, interprocedural data-flow analysis, procedure call, hierarchical-task graph, data-flow analysis, data-dependence analysis.

Class number TP311.11