用计算函数模型增强数据流分析*

胡世亮 臧斌宇 朱传琪

(复旦大学并行处理研究所 上海 200433)

E-mail: byzang@fudan.edu.cn

摘要 精确的数据流分析,需要充分利用条件分支语句的逻辑语义.为了简洁而有效地处理条件分支语句,该文提出了对应于程序段的计算函数模型,在该模型里表示条件分支语句的逻辑语义,并利用文中提出的不确定性消解方法,可以把通常需要逻辑推理来处理的数据流分析问题转化为空间区域之间覆盖关系的判定问题.而这个问题在并行化编译的理论和实践中已有比较成熟的解决办法.

关键词 计算函数模型,数据流分析,条件分支语句,Omega 区域.

中图法分类号 TP314

数据流分析和相关性测试是并行化编译器分析程序潜在并行性的主要方法.实践证明,在并行化科学计算程序方面,它们已经取得了令人鼓舞的成果[1],而依然存在的问题之一是,现有的分析方法还不善于利用条件分支语句的逻辑语义,因而难以充分开发程序的潜在并行性.

并行化编译器主要开发粗粒度并行性.需要并行化的代码段是程序中计算量大、执行时间占总执行时间相当比例的关键代码段.理论推测和实际统计都表明,在应用程序中广泛出现的条件分支语句,在所难免地会影响对于程序关键代码段的并行性分析.如果不能有效地利用条件分支语句提供的语义信息,那么对程序关键代码段的并行性分析就会流于保守,错失并行计算带来的巨大收益.

传统的数据流分析[2,3]侧重分析过程的简捷,对条件分支语句的逻辑语义基本上予以忽略.文献[4]等在分析了程序自动并行化方面依然存在的主要问题后认为,充分利用条件分支语句的逻辑语义势在必行.在文献[5]中用带条件谓词的符号表示法处理该类问题,其实质是 Wolfe 提出的 φ-function 表示法的扩充.此外,还有在文献[6]中提出的 Guarded Array Region 等.它们都试图在条件谓词上实施逻辑推理来导出精确的数据流信息.但遗憾的是,完备的自动化逻辑推理系统,实现起来代价太高,集成到实用化的并行化编译器里并不现实.本文提出的与程序段相对应的计算函数模型,试图为此找到一种简单而有效的解决方案.

本质上,逻辑推理和计算函数模型是同一个问题的求解在不同数学模型下的表示与运算.而计算函数模型相对于逻辑推理的好处在于:概念上的简洁、统一,该模型拓广了数据流分析中的数组区域及其覆盖的概念以统一处理条件的或非条件的定义-引用数据流关系,问题的表述和求解简洁、统一;宜于在现有系统中实现.概念上的简洁、统一使得实现是在既有系统上的拓广扩充,又无需在数据流分析的框架下再引入逻辑推理系统.

本文第 1 部分介绍引入的计算函数模型.第 2 部分阐述该模型下条件分支语句的语义.第 3 和第 4 部分讨论如何利用该模型解决条件分支语句所带来的两类重要问题.最后得出结论.

条件分支语句形式多样,如 C/C++,Java,FORTRAN,Pascal 等程序设计语言中出现的各种 IF 语句、Switch 语句、case 语句等,其中 IF 语句出现得尤为广泛,本文约定:所提到的 IF 语句和条件分支语句含义等价,另外,约定下列标记符号和相关的算符,见表 1.

* 本文研究得到国家自然科学基金(No. 69633030)、国家 863 高科技项目基金(No. 863-306-ZT01-02-01)、教育部科学技术项目基金和国防科技重点实验室基金(No. 97JS76.5.2.JW0701)资助.作者**胡世亮**,1972 年生,硕士生,主要研究领域为并行与分布式计算.**臧斌宇**,1965 年生,副教授,主要研究领域为并行处理,高性能计算.**朱传琪**,1943 年生,教授,博士生导师,主要研究领域为并行处理,高性能计算.

本文通讯联系人:臧斌宇,上海 200433,复旦大学并行处理研究所

本文 1998-09-21 收到原稿,1999-03-09 收到修改稿

Table 1 Conventions for some symbols

表 1 有关的符号约定

Symbol①	Name②	Explanation③
P	Program segment④	A segment of program (in Fortran, C etc.)⑤
$r\|w$	Read/write reference⑥	Read /write reference to a variable (including array)⑦
x,X	Reference, reference set⑧	Reference to a variable (x can be r, r_i, w, w_i)
$R_P\|W_P$	Read/write set⑨	All read/write reference to the inspected variable in the program segment P⑩
NaW_P	Non anti-dependence write set⑪	The non anti-dependence write set of r in the program segment P⑫*
$Set_P(x)$	Reference element set⑬	The set of all array elements or variables referred by x in program segment P (the set is null when x isn't in the program P)⑭ $Set_P(X)=\bigcup_{x\in X} Set_P(x)$

①符号,②名称,③解释,④程序段,⑤一段被编译的程序,⑥读/写引用,⑦对变量(包括数组,下同)的读/写引用,⑧引用,引用集,⑨对变量的引用(x 可以取值为 r,r_i,w,w_i) 读/写集,⑩程序段 P 中对被考察变量所有读/写引用,⑪无反相关写集,⑫程序段 P 中关于 r 的无反相关写集,⑬引用元素集,⑭x 在 P 中引用的所有变量,数组元素的集合(x)不在 P 中时为空集.

1　程序的计算函数模型及其相关的基本概念

在可计算理论里,任何程序都可以视为一个从输入数据空间到输出数据空间的函数映射.设 m,n 分别为输入、输出空间的维数,那么程序对应于映射:$\Psi_P^{(m)}:Z^m\to Z^n$.同理,在数据流分析过程中,对于当前被分析的程序段 P,也可以采用这种函数映射式的程序模型.

1.1　程序的计算函数模型

定义 1.1. 程序段 P 的定义变量集 $\mathrm{In}(P)=\{x_1,x_2,\ldots,x_m\}$ 表示 P 执行时,读引用之前 P 未定义的所有变量的集合,程序段 P 的定义域为:$Dom(P)=D_{x_1}\times D_{x_2}\times\ldots\times D_{x_m}$.

定义 1.2. 程序段 P 的映象变量集 $\mathrm{Out}(P)=\{y_1,y_2,\ldots,y_n\}$ 表示 P 执行时定义,并且被程序其他部分读引用的变量集合,程序段 P 的值域为:$Range(P)=D_{y_1}\times D_{y_2}\times\ldots\times D_{y_n}$.

这里,D_{x_i} 是 x_i 的定义域,D_{y_i} 是 y_i 的定义域.

定义 1.3. 程序段 P 的计算函数模型定义为向量值函数映射:$f_P:Dom(P)\to Range(P)$.

在上述定义中,把数组和数组的子区域(subarray)视为一组分立的变量,带下标的数组元素引用视为一个普通变量.有些程序段有对其输入变量集进行约束的条件谓词 $\varphi(x_1,x_2,\ldots,x_m)$,此时,程序段 P 的定义域应该是:

$$Dom(P)=\{(x_1,x_2,\ldots,x_m)\mid\varphi(x_1,x_2,\ldots,x_m)=\mathrm{True}\}.$$

为了在上述计算函数程序模型的基础上进行推理,我们还假设了下面的程序执行方式.

确定性执行假设　对于程序段 P 定义域中的任何确定的一点(对应着程序段 P 的一次实例执行):$(x_1,x_2,\ldots,x_m)\in Dom(P)$,程序段 P 的执行是完全确定的,这种确定性包括:

P 的 $\mathrm{In}(P)=\{x_1,x_2,\ldots,x_m\}$ 和 $\mathrm{Out}(P)=\{y_1,y_2,\ldots,y_n\}$ 确定;

P 计算结束后 $\mathrm{Out}(P)=\{y_1,y_2,\ldots,y_n\}$ 中的值也是确定的.即 P 是确定性的程序,f_P 是严格数学意义下的函数映射;

条件分支语句的转向确定.

确定性执行假设对于我们后面的推理有着基本的重要性,并且常见的程序设计语言一般都满足上述假设.

1.2　不确定性概念的引入

定义 1.4. 如果$\exists$ $A\subset\mathrm{In}(P)$,$A\neq\varnothing$,当 A 中的变量由于不同的执行实例而有不同取值时,$\mathrm{In}(P)$ 或 $\mathrm{Out}(P)$

* 这里定义:对于程序段 P 中的读引用 r,称 P 中写引用 w 是关于 r 的无反相关写,如果 r 与 w 在 P 中无反相关.P 中所有关于 r 的无反相关写构成 P 中关于 r 的无反相关写集 NaW_P.

也在该实例的后续执行中取值为不同的变量集,则称程序段 P 是数据流不确定的.而 A 中的变量是造成这种不确定性的变量,在编译器中需要进行符号分析,我们称之为关键值变量或符号变量.

我们称程序段 P 的数据流不确定现象为流不确定性,流不确定性主要由条件分支语句、指针别名等产生;也会由于某些关键值的不确定、数据引用以及 DO 语句的迭代范围等导致.其直观意义是,程序段 P 对于某些变量是否进行了读写由于是在一定逻辑条件下进行的,或读写的范围由于某些变量值的不明确而产生数据流不确定现象.

当数据流确定时,如果对于编译器来说,变量定义的具体数值不明确,我们则称之为值不确定性.值不确定性可以由表达式、条件分支语句、程序变量的输入值等导致.其直观含义是:程序段 P 的读写引用区域是确定的,但程序段 P 中具体数值计算的映射关系不明确.

在确定性执行假设下,程序数据流信息的不确定性源自程序输入数据的不确定(可以是 $Dom(P)$ 中的任何一点),或计算结构的复杂度超出编译器的分析能力.

任何程序段 P 都可以视为是某个可计算函数 f 的实现,编译器的任务是产生一个同 f 等价的映射 $f_{\text{optimized}}$:它和 f 具有在 $Dom(P)$ 内完全相同的值对应关系,只是 $f_{\text{optimized}}$ 必须具有更加适合于底层计算模型的计算结构以获取高性能.编译器不是映射 f 的具体实施,不关心具体的数值计算过程,所以值不确定性对编译器无足轻重;但流不确定性意味着变量值在程序中的传递关系不明确(计算函数的复合关系不明确),是编译器对于程序计算结构的识别和理解不明确.因此,流不确定性对于编译器是关键的.编译器在对程序的一个局部进行并行性的分析与变换时,需要识别,以消解编译时刻可以确定的流不确定性,以保持在本质的计算约束下并行化和优化程序.否则,就只能在不确定性的可能范围里作出保守的假设.

1.3 条件谓词在计算函数模型下的语义

在计算函数模型下,程序段 P 中任何条件分支语句的条件谓词,在经过一定的变量替换后都对应着 $Dom(P)$ 中一个特定的子区域(在无任何条件约束的情况下为 $Dom(P)$).

设 $Predicate(t_1,t_2,\ldots,t_k)$ 是条件分支语句中的逻辑条件,$t_1,t_2,\ldots,t_k$ 是条件中直接出现的变量,那么 $Predicate(t_1,t_2,\ldots,t_k)=\text{True}$ 确定了 $D_T=D_{t_1}\times D_{t_2}\times\ldots\times D_{t_k}$ 空间中的一个区域 Ω_T,由程序中从 P 的定义变量到参数变量 $t_1,t_2,\ldots,t_k$ 的一系列赋值语句、表达式等确定的映射关系:$\varphi_t(x_1,x_2,\ldots,x_m)\to(t_1,t_2,\ldots,t_k)$ 可知,D_T 是 $Dom(P)$ 的一个导出参数空间.$\Omega_P=\varphi^{-1}(\Omega_T)$ 表示在 P 的定义域中,该条件分支执行的程序状态空间区域.而 Ω_T 表示是在参数空间 D_T 中.这里的 $\Omega_P=\varphi^{-1}(\Omega_T)$ 有如下表示:

$$\Omega_P=\{(x_1,x_2,\ldots,x_m)\mid(x_1,x_2,\ldots,x_m)\in Dom(P);\varphi(x_1,x_2,\ldots,x_m)=(t_1,t_2,\ldots,t_k)\in\Omega_T\}.$$

事实上,谓词 $p(x_1,x_2,\ldots,x_n)$ 是程序状态空间中的特定空间区域或点集 Ω_P 的特征函数.

$$p(x_1,x_2,\ldots,x_n)\Leftrightarrow(x_1,x_2,\ldots,x_n)\in\Omega_P,\forall\ (x_1,x_2,\ldots,x_n)\in Dom(P).$$

定义 1.5. 以程序段 P 中的条件谓词 $Predicate(t_1,t_2,\ldots,t_k)$ 作为特征函数,在 P 的定义域 $Dom(P)=D_{x_1}\times D_{x_2}\times\ldots\times D_{x_m}$ 中确定的空间区域 Ω_P 以及所有表示在 $Dom(P)$ 的导出空间 $D_T=D_{t_1}\times D_{t_2}\times\ldots\times D_{t_k}$ 中的区域 Ω_T 称为该谓词的 Omega 区域,记为 $\Omega_T^{\text{Predicate}}$,在不导致歧义时,简化为 $\Omega_{\text{Predicate}}$.

引理 1.1. 设程序段 P 中的逻辑谓词 $p(x_1,x_2,\ldots,x_n)$ 确定的 Omega 区域为 Ω_p,则:

1. $\neg p(x_1,x_2,\ldots,x_n)\Leftrightarrow(x_1,x_2,\ldots,x_n)\in\overline{\Omega}_P-Dom(P)-\Omega_P$,
2. $p_1(x_1,x_2,\ldots,x_n)\wedge p_2(x_1,x_2,\ldots,x_n)\Leftrightarrow(x_1,x_2,\ldots,x_n)\in\Omega_{p_1}\cap\Omega_{p_2}$,
3. $p_1(x_1,x_2,\ldots,x_n)\vee p_2(x_1,x_2,\ldots,x_n)\Leftrightarrow(x_1,x_2,\ldots,x_n)\in\Omega_{p_1}\cup\Omega_{p_2}$,
4. $p_2(x_1,x_2,\ldots,x_n)\to p_1(x_1,x_2,\ldots,x_n)\Leftrightarrow\Omega_{p_1}\supseteq\Omega_{p_2}$,
5. $p_1(x_1,x_2,\ldots,x_n)\leftrightarrow p_2(x_1,x_2,\ldots,x_n)\Leftrightarrow\Omega_{p_1}=\Omega_{p_2}$,
6. $p(x_1,x_2,\ldots,x_n)\to\bigvee_{i=1}^{n}p_i(x_1,x_2,\ldots,x_n)\Leftrightarrow\bigcup_{i=1}^{n}\Omega_{p_i}\supseteq\Omega_p$.

证明是平凡的,略.

引理 1.2. 设 $\mathscr{P}$ 是关于程序变量的逻辑谓词集,$\Omega_{\mathscr{P}}$ 是以 $\mathscr{P}$ 中的谓词为特征函数生成的 Ω 区域的集合,则点集运算系统$[\Omega_{\mathscr{P}};^{-},\cap,\cup]$同态于逻辑运算系统$[\mathscr{P};\neg,\wedge,\vee]$.

由引理 1.1 易知,$\Omega_{\mathscr{P}}$ 的生成映射 $\varphi:\mathscr{P}\to\Omega_{\mathscr{P}}$ 是其上的同态映射,证明略.

引理 1.1 和引理 1.2 表明,在条件分支语句中,逻辑谓词对应的 Omega 区域,其相互间的覆盖关系对应着逻辑谓词之间的逻辑关系,因此,计算函数模型的引入,可以把逻辑推理的问题变换为空间区域之间覆盖关系的判定问题.

逻辑谓词及其对应的 Omega 区域在表示形式上是一致的,都是关于程序变量的等式或不等式组等.区别在于作用于其上的运算系统是不同的,区域覆盖的方法更容易与现有技术融和,更易于实现.判定 Omega 区域之间的覆盖关系时,没有必要把变量全部替换为 P 的定义变量,而只需把两个 Omega 区域划到同一个参数变量空间即可.在文献[5]中提出了一个逐步反向替代的变量替换方法.对实用程序,特别是科学计算类的程序的分析统计表明,多数条件分支语句的逻辑条件谓词确定了其参数空间里的线性凸区域,或线性凸区域中按照一定规律排布的整数点集.而针对这种区域、点集的集合运算处理方法(如线性不等式组等)在并行化编译器的实践中已经比较成熟.

在 FORTRAN 程序及 C 语言程序中,这里的程序段 P 可以是程序中的各种实体:整个程序、某个子程序、某块代码、DO 的循环体,甚至是一个语句或表达式.这种统一的看法有助于推出一个统一的处理方法,它既可以具有局部有效性,又可以进行全局的通盘筹划处理.

2 计算函数模型下条件分支语句的语义

条件分支语句的一般形式:

if	if
[] $P_1\to B_1$;	[] $\Omega_{P_1}\to B_1$;
[] $P_2\to B_2$;	[] $\Omega_{P_2}\to B_2$;
... or	...
[] $P_n\to B_n$;	[] $\Omega_{P_n}\to B_n$;
fi	fi

由于 $B_1,B_2,\ldots,B_n$ 执行的不确定性导致了程序对数据读写的不确定性,这种不确定性由各个可执行语句体 B_i 的前置谓词 P_i 决定.P_i 的引用区域是完全确定的,不会导致不确定性.

2.1 条件分支语句对变量定义的语义作用

设当前分析的程序段 P,条件分支语句对其中变量定义产生的语义作用可以有如下几种.

· **值不确定性定义(写)** 参见下面图 1 和图 2 中的示例,作为一个整体,程序段的所有读写区域是确定的,但变量定义的数值不确定.此时,IF 语句对于变量定义的语义作用是计算表达式的一部分,主要意图是在不同条件下赋予变量不同的数值.如果条件分支语句的此类语义作用不影响程序的计算结构(比如其产生的变量值出现在其他 IF 语句的条件表达式、DO 语句的上下界表达式或数组下标表达式中)时,可以把它作为一种表达式来处理.而当条件分支语句下的定义结果影响程序的计算结构时,则该值不确定性定义是符号变量定义,需要进行符号分析.我们采用下面的标记表示其语义:$\Omega_P\xrightarrow{w}ExpVal_i$,其中 Ω_P 是条件谓词 p 的 Omega 区域,w 表示是写/定义,$ExpVal$ 表示相应的计算表达式.

```
Smax=A(1)
DO I=2,N
    IF (Smax<A(I)) THEN
        Smax=A(I)
    ENDIF
ENDDO
Smax=max_{i=1}^{n}(A(i))
```

Fig. 1
图 1

```
DO 103 K=1,7,1
    QIN(K)=QN1(I,K)
    IF (QN1(I,K).LT..0) THEN
        QIN(K)=.0
    ENDIF
ENDDO
QIN(K)=max(QN1(K),0.0)
```

Fig. 2
图 2

· **流不确定性定义(条件写)** 参见下面的示例.IF 语句决定的是变量,特别是数组区域(array region)在一

定条件下的写区域.逻辑条件成立的不确定导致数据定义和引用关系的不确定,其中的逻辑语义暗示了程序计算结构的约束关系.相应的语义可以标记为:$\Omega_p \xrightarrow{w} Set_p(x)$.

条件分支语句产生的流不确定性示例如下:

```
[1]  DO K=2,5,1
[2]      IF (RS(4+K).LE.CUT2) THEN
[3]          RL(4+K)=SQRT(RS(4+K))
[4]      ENDIF
[5]  ENDDO
[6]  IF (KC.EQ.0) THEN
[7]      DO K=11,14,1
[8]          FF((-5)+K)=FF((-5)+K)+AB2*EXP(-B2*RL(K-5))/RL((-5)+K)
[9]      ENDDO
[10] ENDIF
```

整个程序段是否定义、引用了 $RL(6:9)$是不确定的.

流不确定性一定是由值不确定性引发,并且值不确定性定义和流不确定性定义是相对于当前程序段 P 的,对于程序段 P 是流不确定性的定义在一个更大的程序范围内可能会成为值不确定性定义,反之亦然.区分条件分支语句的上述两种语义是出于数据流分析过程的需要,并且区别这两种条件分支语句对于变量定义的语义作用不能简单地通过对条件分支语句的词法分析和语法分析得到,而需要有在一定范围内的数据流分析,以确定其计算语义.

2.2 条件分支语句对变量读引用的语义作用

条件分支语句对于变量读引用的语义作用是,该读引用是在一定逻辑条件下进行的,这里我们称之为条件引用(读).相应的逻辑条件确定的 Omega 区域内是该读引用发生的程序状态集,并且也是我们用来化解定义-引用之间数据流不确定性的关键条件.条件读的语义在计算函数模型下表示为

$$\Omega_P \xrightarrow{r} SymbolicVariable; \qquad \Omega_P \xrightarrow{r} Set_p(x).$$

2.3 条件分支语句的处理

对于由条件分支语句导致的单纯的值不确定性定义予以忽略.由于在科学计算程序中有相当一部分条件分支语句是值不确定性定义,这种忽略可以大大减轻编译器的计算量.而对于条件分支语句下的符号变量定义,则应设法推测其值的情况.此时,值的信息会导出流的信息.本文第 4 部分介绍如何利用计算函数模型辅助符号分析来处理其中的逻辑条件.

流不确定性定义的处理.利用条件读中的逻辑条件进行定义-引用数据流分析,消解编译时刻可以明确的数据流的不确定性.下面所证明的覆盖定理将利用前面介绍的 Omega 区域之间的覆盖关系来解决这个问题.需要指出的是,许多场合下的数据流不确定性是不能在静态得以明确的,本质上,它们是动态确定的数据流关系.

3 用计算函数模型消解流的不确定性

3.1 流不确定性消解需要解决的问题

在数据流分析中确定读引用的数据源以得出程序段的数据流信息是许多并行化、优化变换的前提.但是,由于条件分支语句限定下的条件定义没有 kill 掉其他定义点的写,甚至是该定义点以前实例的定义,由此导致了读引用对于同一存储变量的读取可能读到程序段 P 当前实例之外的定义点,产生流不确定性.忽略这种不确定性意味着无条件地接受其他数据源的可能,分析结果趋于保守.

在计算函数模型的框架下可以把问题表述为:对于一个读引用 $r:\Omega_{pr} \xrightarrow{r} Set_p(r)$(表示在计算函数模型下,下同)以及可能为 r 提供数据源的定点集 $W=\{w:\Omega_{pw} \xrightarrow{w} Set_p(w)\}$,如何确定 W 和 r 之间的数据流关系.

下面的讨论针对特定的数组 A,标量的情形可以简单地推出.程序段 P 可以是任何复杂嵌套情形下的结构

化程序段.

3.2 流不确定性消解原理

定义 3.1. 写覆盖读. 对于程序段 P 中的一对读写 r,w(或写集 W),如果在 P 的任何执行实例中:写的执行在读 r 之前执行并且 r 读到的数据集都被写 w(或写集 W)的定义数据集所包含,换句话说,写 w(或写集 W)完全可以满足读 r 的数据引用需求,我们则称写 w 覆盖了读 r(或写集 W 覆盖了读 r),记为:

$$w\xrightarrow{f}r \text{ 或 } W\xrightarrow{f}r.$$

定理 3.1. 如果程序段 P 中的一对读写 r,w:$w\in NaW_P^r$,满足下列条件:

$$\Omega_r\subseteq\Omega_w \quad \text{and} \quad Set_P(r)\subseteq Set_P(w),$$

那么,写 w 覆盖了读 r,$w\xrightarrow{f}r$.

证明:由于 $\Omega_r\subseteq\Omega_w$,当程序段 P 执行读 r 时,写 w 也一定被执行到,由 $Set_P(r)\subseteq Set_P(w)$ 可知,w 和 r 之间一定存在相关性,可能是流相关,也可能是反相关,但由题设:$w\in NaW_P^r$,w 和 r 之间存在的一定是流相关性,即写完全有能力向读 r 提供数据,因此:$w\xrightarrow{f}r$. □

如果 $w\xrightarrow{f}r$,并且 w 输出相关于任何同 r 有流相关的写,w 是 r 的确切数据源.

数据集空间 DataSpace 定义为程序中由数据变量的存储单元组成的存储空间. 标量 S 的 DataSpace 是标量 S 所代表的一个存储单元空间;数组 A 的 DataSpace 是数组 A 所代表的 $dim(A)$ 维存储空间,其上下界由数组说明提供;程序段 P 的 DataSpace 由程序段 P 的所有变量,包括标量和数组的全体,组成的存储空间. 下面的讨论中主要涉及数组的 DataSpace.

为了导出写集覆盖读的充要条件,我们需要考虑 Omega 区域和数组引用区域的笛卡积. 它代表程序状态空间和程序数据集空间的笛卡积:$Dom(P)\times DataSpace$ 中的一个特定区域:$\Omega_x\times Set_P(x)$. 其直观意义是:程序段 P 在满足 Ω_x 的程序状态下引用数据区域 $Set_P(x)$.

定义 3.2. 程序段 P 中引用 x 的状态-数据引用区域,简称 Φ 区域,定义为:

$$\Phi_P^x=\Omega_P^x\times Set_P(x).$$

Ω_P^x 表示在程序段 P 范围内约束引用 x 的 Omega 区域,在不产生歧义时简记为 Ω_x,Φ_P^x 也记为 $\Phi_P(x)$. 定义:$\Phi_P(X)=\bigcup_{x\in X}\Phi_P(x)$.

定理 3.2(覆盖定理). 对于程序段 P 中的读 r 和写集 W:$W\subseteq NaW_P^r$,$W\xrightarrow{f}r$ 的充要条件是:读的状态-数据引用区域被写集的状态-数据引用区域之并所覆盖:$\Phi_P^r\subseteq\bigcup_{w\in W}\Phi_P^w$,即$(\Omega_r\times Set(r))\subseteq\bigcup_{w\in W}(\Omega_w\times Set_P(w))$.

证明:充分性:$\Phi_P^r\subseteq\bigcup_{w\in W}\Phi_P^w$ 推出 $W\xrightarrow{f}r$.

当程序段 P 任何一次执行到读 r 时,其所对应的程序状态为:$s\in\Omega_r$. 它对存储区域任何一点 $X\in Set_P(r)$ 的引用必定满足下列条件:

$\exists w\in W$,使得 $s\in\Omega_w$ 且 $X\in Set_P(w)$(否则与题设:$\Phi_P^r\subseteq\bigcup_{w\in W}\Phi_P^w$ 矛盾). 由于 $X\in Set_P(r)\wedge X\in Set_P(w)$,$w$ 和 r 之间一定存在着相关性,可能是流相关,也可能是反相关,但由题设:$w\in W\subseteq NaW_P^r$,则 w 和 r 之间存在的一定是流相关. 由写覆盖读的定义,充分性得证.

必要性:$W\xrightarrow{f}r$ 推出 $\Phi_P^r\subseteq\bigcup_{w\in W}\Phi_P^w$.

反证法:设$(s,X\in\Phi_P^r)$但$(s,X)\notin\bigcup_{w\in W}\Phi_P^w$,则由覆盖的定义,当程序段 P 以 s 进入且引用点 X 时,W 无法覆盖 r,与题设 $W\xrightarrow{f}r$ 不符. 可见必要性成立. □

推论 1. 如果程序段 P 中的一对读写 r,w:$w\in NaW_P^r$,且满足下列条件:

$$(\Omega_r\times Set_P(r))\subset(\Omega_w\times Set_P(w)),$$

那么,写 w 覆盖了读 r,$w\xrightarrow{f}r$.

推论 2. 设 r 为程序段 P 中的读,如果 $\Phi_P^r\subseteq\bigcup_{w\in NaW_P^r}\Phi_P^w$,那么,$r$ 为 P 中的非暴露读.

推论 3. 对于程序段 P 中的读集 R 和写集 W:$W\xrightarrow{f}R$ 的条件是:

$\forall r\in R,\exists W^*\subseteq W\wedge W^*\subseteq NaW_P\wedge W^*\xrightarrow{f}r$,其中 $W^*\xrightarrow{f}r$ 的充要条件由覆盖定理确定:$\Phi_P^r\subseteq\bigcup_{w\in W^*}\Phi_P^w$.

推论 1 是定理 3.1 的另外一种表示形式,推论 2 应用在数组私有化判定过程中,推论 3 是覆盖定理的一种广义形式.

由覆盖定理,我们可以确定程序段 P 中特定的定义点集 $W=\{w\mid\Omega_{pw}\xrightarrow{w}Set_P(w)\}$ 能否满足 P 中读引用 r:$\Omega_{pr}\xrightarrow{r}Set_P(r)$ 的数据引用需求.

4 用计算函数模型辅助符号分析

4.1 条件分支语句对于符号分析的影响

符号分析对于精确的数据流分析和相关性测试十分重要.符号变量一般是标量,确定这些变量的数值可以使编译器分析出更加精确的数据流信息.

在计算函数模型下,条件分支语句里定义的符号变量,其值不确定性可以表示为:

$$Symbol=\{\Omega_{p1}\xrightarrow{w}ExpVal_1,\ldots\ \Omega_{pn}\xrightarrow{w}ExpVal_n\ \Omega_{\text{other}}\xrightarrow{w}ExpVal_{\text{other}}\}.$$

由确定性执行假设可知,$\Omega_{p1},\ldots,\Omega_{pn},\Omega_{\text{other}}$ 是 $Dom(P)$ 的一个划分.

相应的要解决的问题是:对于一个关键值 $Symbol$ 的读引用 $R:\Omega_{pr}\xrightarrow{r}Symbol$ 以及 $Symbol$ 可能的取值:

$$Symbol=\{\Omega_{p1}\xrightarrow{w}ExpVal_1,\ldots\ \Omega_{pn}\xrightarrow{w}ExpVal_n\ \Omega_{\text{other}}\xrightarrow{w}ExpVal_{\text{other}}\},$$

如何确定 R 实际读取的 $Symbol$ 变量的值.

4.2 用计算函数模型消解数值的不确定性

事实上,化解值不确定性和消解流不确定性在原理上是一致的,我们可以得出下面的结论.对于程序段 P 中一个对关键值 $Symbol$ 的读引用 $R:\Omega_{pr}\xrightarrow{r}Symbol$,$R$ 如果能够读到 $ExpVal_i$,则一定满足 $\Omega_{pr}\cap\Omega_{pi}\neq\varnothing$.特别地,如果 $\Omega_{pr}\subseteq\Omega_{pi}$,则 R 读到的一定是 $ExpVal_i$,如果 $\Omega_{pr}\subseteq\bigcup_{i\in D}\Omega_{pi}$,则 R 读到的一定是 $\{ExpVal_i\mid i\in D\}$ 中的某个数值.

证明类似于流不确定性的消解,略.

5 结 论

任何试图利用条件分支语句的逻辑语义来得到精确数据流信息的方法,本质上都是在不同的模型下,对逻辑条件进行表示和运算.在计算函数模型的框架下,条件分支语句的语义可以自然而简洁地表示,其形式为:$\Omega_P\xrightarrow{x}Set_P(x)$ 或 Φ 区域,是数组引用区域在概念上的一个拓广;而条件分支语句语义作用下的相应数据流信息也可以通过读写引用的 Ω 区域或 Φ 区域的运算得以明确.这里的表示和计算方式与目前数组数据流分析采用的方式基本一致,因此实现起来只需对已有系统进行扩充,无需引入一套推理系统.

运用本文提出的不确定性消解原理,通过手工形式化变换 PERFECT Club Benchmarks,SPEC95fp 等测试程序包里的实用程序表明,这里提出的方法简洁有效.下一步的工作是进行算法化,以便在并行化编译器中实现.

参考文献

1 Zhu Chuan-qi, Zang Bin-yu, Chen Tong. An automatic parallelizer. Journal of Software, 1996,7(3):180~186
(朱传琪,臧斌宇,陈彤.程序自动并行化系统 AFT.软件学报,1996,7(3):180~186)

2 Mary W Hall, Murphy B R, Amarasinghe S P *et al*. Interprocedural analysis for parallelization. In: Huang C-H *et al* eds. Proceedings of the 8th International Workshop on Languages and Compilers for Parallel Computing. Columbus, Ohio: Springer-Verlag, 1995. 61~74

3 Beatrice Creusillet, Francois Irigoin. Interprocedural array region analysis. In: Huang C-H *et al* eds. Proceedings of the 8th International Workshop on Languages and Compilers for Pa·allel Computing. Columbus, Ohio: Springer-Verlag, 1995. 46~60

4 Peng Tu, David Padua. Automatic array privatization. In: Proceedings of the 6th International Workshop on Languages and Compilers for Parallel Computing. Springer-Verlag, 1993. 500～521

5 Peng Tu, David Padua. Gated SSA-based demand-driven symbolic analysis for paralleling compilers. In: Michael Wolfe, Denis Nicole *et al* eds. Proceedings of the 1995 International Conference on Supercomputing. Bacelona: ACM Press, July 1995. 414～423

6 Trung Nguyen, Gu Jun-jie, Li Zhi-yuan. An interprocedural paralleling compiler and its support for memory hierarchy research. In: Huang C-H *et al* eds. Proceedings of the 8th International Workshop on Languages and Compilers for Parallel Computing. Columbus, Ohio: Springer-Verlag, 1995. 90～104

Enhancing Dataflow Analysis with Computation Function Model

HU Shi-liang ZANG Bin-yu ZHU Chuan-qi

(*Institute of Parallel Processing Fudan University Shanghai* 200433)

Abstract A precise dataflow analysis should effectively exploit the semantic information presented by conditional branch statements. Most traditional systems, however, either ignore the logical conditions, or try hard to handle logical conditions with logical reasoning which is rather difficult and infeasible for paralleling systems. With the aim to solve this problem efficiently and effectively, the authors introduce the computation function model and then successfully convert the logical reasoning problems into problems of deciding the coverage relationship of Omega regions which are substantially more feasible to be solved by contemporary paralleling compilers.

Key words Computation function model, dataflow analysis, conditional branch statement, omega region