# 基于路径信息的谓词关系分析

沈 立,王志英,鲁建壮
(湖南长沙国防科技大学计算机学院 6 队,湖南长沙 410073)

**摘 要**: 谓词执行技术能够将多个基本块合并为一个超块,扩大指令调度范围,开发更多的指令级并行. 但它也给编译优化带来一些新问题,使得传统的编译器在进行指令调度或寄存器分配等优化工作时往往只能得到保守的结果. 受所用谓词表示方法的限制,目前的谓词关系分析方法必须首先重构控制流信息,这就影响了编译优化的效果和效率. 本文提出了一种基于路径编码的谓词表示方法,将谓词定义信息转换为等价的域编码,并以此为基础实现了一个全局谓词关系分析子系统. 模拟结果表明,该子系统能够准确高效地实现全局谓词关系分析,在保证编译效率的同时,提高了编译优化的效果.

**关键词**: 谓词执行; 谓词分析; 路径; 域
**中图分类号**: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2004) 02-0191-05

# Predicate Analysis Based on Path Information

SHEN Li, WANG Zhi-ying, LU Jian-zhuang
(*National University of Defense Technology, Computer School, Changsha, Hunan* 410073, *China*)

Abstract: Several basic blocks could be merged into a hyperblock in predicated execution so that instructions could be scheduled on a larger scope and more instruction level parallelism could be extracted. But it also brings some new challenges to traditional optimizers without predicate analysis, which can only yield conservative results. But unfortunately, limited by predicate representations, current methods need reconstructing control flow information before analyzing predicate relationships, which limits the effectiveness and efficiency of optimization. A new representation based on path information is proposed in this paper, which can convert predicate definitions into equivalent domain codes. And a global predicate relationship query system is constructed based on it. Experiment results indicated that precise and efficient global predicate analysis could be achieved with this system and the performance of optimized codes can also be improved with little impact on compilation efficiency.

Key words: predicated execution; predicate analysis; path; domain

## 1 引言

### 1.1 谓词执行

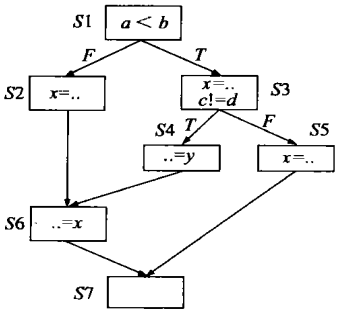
谓词执行技术[1,2]为每一条指令增加了一个布尔类型的源操作数(谓词),谓词的值决定了该指令应正常执行还是被转换为空操作. 目前的谓词执行技术几乎都以 HPL PlayDoh 体系结构模型[2]为基础实现,该模型将谓词保存在一组 1 位的谓词寄存器中. 将普通代码转换为谓词代码的编译技术称为条件转换[3]. 图 1 给出了一段代码及其经过条件转换后得到的谓词代码,其中子图 1(*a*) 为控制流图,图 1(*b*) 为相应的非谓词代码,图 1(*c*) 为转换后得到的谓词代码,图 1(*d*) 则表示谓词代码的并行指令集合. 谓词执行有许多优点. 首先,由于消除了代码中的分支指令,原先存在的控制相关也被转换为相对于分支条件的数据相关,多个基本块被合并为体积更大的超块(hyperblock),扩大了指令调度的范围,为开发出更多 ILP 提供了前提. 值得注意的是,此时基本块体积的增加是对代码体积影响很小的前提下实现的,因此与代码复制相比,后者更具吸引力. 其另一个优点在于,条件转换能够消除一部分难以预测的分支,从而提高分支预测准确率,并减少运行时因分支预测失败造成的性能损失.

从图 1(*d*) 可以看出,条件转换能够有效提高指令级并行. 研究结果也表明了这一点:合并多个控制流能够将指令级并行增加到原来的 3 倍[4],能够有效地提高其他代码变换技术(如软流水和模块调度)的性能,而且谓词执行技术能够平均消除 27 %的分支以及 56 %的分支预测错误[5,6].

### 1.2 谓词分析

尽管谓词执行技术具有相当的性能潜力,但只有借助强大的编译系统才能将其实现. 编译优化的各个阶段,包括指令调度和寄存器分配,都必须进行精确高效地谓词关系分析,才能生成高质量的目标代码. 以图 1(*c*) 为例,指令 $I_7$ 与 $I_9$ 分别

收稿日期:2003-03-26;修回日期:2003-09-10
基金项目:863 高技术研究发展计划(No. 2001AA111050)




(a) 控制流图

```
S1:1=cmp(a < b)
   branch 1, S3
S2: x=..
   branch S6
S3: x=..
   m=cmp (c!=d)
   branch m, S4
S5: x=..
   branch S7
S4: ..=y
S6: ..=x
S7:
```

(b) 非谓词代码

```
S1: p,q=cmpp.uc.un (a < b)          11
   t=cmpp.uc(a < b)                 12
   x=..                    if p     13
   x=..                    if q     14
   r,s=cmpp.uc.un (c!=d)   if q     15
   t=cmpp.on (c!=d)        if q     16
   x=..                    if s     17
   ..=y                    if r     18
   ..=x                    if t     19
S7:
```

(c) 谓词代码

```
p,q=cmpp.uc.un(a<b);t=cmpp.uc(a<b)
x=..if p;  x=..if q;  r,s=cmpp.uc.un(c!=d)if q;  t=cmpp.on(c!=d) if q;
x=..if s;  ..y if r;  .. =x if t;
```

(d) 并行指令集合

图 1 谓词代码实例

对变量 $x$ 进行写操作和读操作. 按照传统的分析方法,它们不可能同时执行. 实际上,由于它们所对应的谓词 $s$ 和 $t$ 不可能同时为真,因此其执行结果不可能同时被确认,它们可以同时执行. 而指令 $I_3$ 和 $I_4$ 都对变量 $x$ 进行写操作,按照传统方法,它们应各自占用一个物理寄存器. 但由于它们所对应的谓词 $p$ 和 $q$ 不可能同时为真,因此它们能够共享同一个物理寄存器. 如果两个谓词的值不可能同时为真,则称其互斥(disjoint). 另一个需要分析的谓词关系为蕴含(imply). 对于两个谓词 $p$ 和 $q$,如果 $p$ 为真时 $q$ 必定为真,$p$ 为假时 $q$ 可以为真也可以为假,则称 $p$ 蕴含 $q$,记作 $p \Rightarrow q$. 分析蕴含关系能够检测伪数据相关. 以图 2 为例,假设 $p \Rightarrow q$,则第三条指令可以修改为: $x = 7$ if (not $q$). 这样,第一条和第三条指令之间没有冲突,可以并行执行.

```
x = 4        if p
branch L     if q
x = 7
```

图 2 谓词蕴含关系

与传统编译器相同,谓词关系分析的范围越大,越有助于提高编译优化的效果. 因此本文将探讨如何在全局范围内精确高效地分析这两种谓词关系.

### 1.3 相关工作

由 1.2 节可知,谓词关系分析主要根据谓词定义信息判断两个谓词的值之间是否存在互斥和蕴含关系. 目前一些具有代表性的谓词分析技术主要包括: IMPACT 使用谓词层次图(Predicate Hierarchy Graph, PHG)[4]跟踪一个超块中所有布尔表达式的定义和使用, PHG 能够准确地描述谓词之间的层次关系,但无法描述其它结构的谓词关系. 而文[8,9]中则使用谓词关系分割图(Predicate Partition Graph, PPG)跟踪超块内的局部谓词关系,并为数据流分析提供相应的信息. 它们首先依照代码中的谓词定义指令谓词分割图,并根据谓词之间的分割关系判断它们之间是否存在互斥、蕴涵等关系. 但 PPG 只能精确描述表示直接通过条件转换得到的谓词之间的关系,而无法有效地表示通过其他方法如循环剥落和模块调度生成的谓词,只能将这些方法产生的谓词转换为一种近似的表示. 这样做的结果是对于一组谓词定义指令可能会生成几个精度不同的谓词分割图,从而会影响谓词分析甚至数据流分析的效果. 上述方法的最大问题是每次需要分析谓词关系的时候都必须首先根据谓词运算指令重构控制流信息,增大了编译优化的复杂度和开销. 造成这一问题的主要原因是没有有效的谓词表示方式. 谓词分析需要根据谓词定义信息判断谓词之间的关系,而现有的工作为减少空间开销,在条件转换后谓词寄存器只保存了谓词的值,不得不在谓词分析时通过数据流分析重新生成谓词定义信息,这就增加了实现的复杂度和开销. 为解决这一问题,本文提出了一种基于路径信息的谓词表示方法,通过域编码表示谓词定义信息. 域编码在条件转换的同时生成,由于保存域编码所需的空间开销很小,可以在整个编译后端统一维护保存域编码信息,从而省略谓词定义信息的重构过程,提高谓词分析的效率. 以此为基础我们实现了一个全局谓词关系查询系统,测试数据表明该系统能够有效地进行谓词关系分析并提高目标代码的质量.

## 2 基于路径信息的谓词表示

### 2.1 基本定义

由 1.3 节中的分析可知,决定谓词关系分析效果和效率的关键在于采用何种方法表示谓词定义信息,本节主要讨论如何利用路径信息表示谓词定义信息. 以图 1(a) 为例,一共有 3 条以 $S_1$ 为起点 $S_7$ 为终点的路径,分别是: $S_1S_2S_6S_7$、$S_1S_3S_4S_6S_7$ 和 $S_1S_3S_5S_7$. 显然,运行时当执行基本块 $S_i$ ($1 \le i \le 7$) 所在的路径时, $S_i$ 也将被执行. 这说明基本块谓词的值与它所在的路径密切相关.

**定义 1** 控制流图 $G = <N, E>$ 是一个有向图, $N$ 为结点集, $G$ 中的一个结点代表一个基本块,而边集 $E = \{ <S_i, S_j> \mid S_j$ 是 $S_i$ 的后继基本块$\}$. $G$ 中每个结点的出度最多为 2.

为了说明控制流从结点 $S_i$ 进入后继结点 $S_j$ 的条件,我们为 $G$ 上的每条边分配一个标记. 如果结点 $S_i$ 与 $S_j$ 之间没有分支,则其标记为空;否则,如果分支条件为真时控制流进入 $S_j$ 则标记为 $T$,否则标记为 $F$.

**定义 2** 结点 $S$ 是路径 $P$ 中的一个结点,则 $S$ 在路径 $P$ 中的编码(简称路径编码) $PC(S)$ 可递归地定义为:

如果 $S$ 是路径 $P$ 的第一个结点,则 $PC(S) = nil$;

否则假设 $S$ 的路径 $P$ 上的前驱结点为 $R$;

如果沿路径 $P$,边 $RS$ 无标记,则 $PC(S) = PC(R)$;

如果沿路径 $P$,边 $RS$ 标记为 $T$,则 $PC(S) = PC(R) \| 1$;

如果沿路径 $P$,边 $RS$ 标记为 $F$,则 $PC(S) = PC(R) \| 0$.

这里'‖'表示串的连接操作.

**定义 3**　结点 $S \in N$, $S$ 的域 $D = \{P \mid P$ 是一条路径, $S$ 是 $P$ 中的一个结点$\}$. $S$ 的域 $D$ 的编码(简称域编码) $DC(S) = \{PC(S) \mid P \in D\}$.

```
Domain Encode(G)
1:将 DC 中的每个元素设为空集
2:在条件转换过程中,当处理结点 S 时
3:if S 在控制流图中没有前驱结点
4:    DC(S) = {nil}
5:else{
6:    设 S 的所有后继结点集合为 S_d
7:    for S_d 中的每个元素 A
8:    for D_s 中的每个路径编码 PC(S){
9:        1. 如果边 SA 没有标记,则 PC(A) = PC(S)
10:       2. 如果边 SA 标记为 T,则 PC(A) = PC(S) ‖1
11:       3. 如果边 SA 标记为 F,则 PC(A) = PC(S) ‖0
12:       DC(A) = DC(A) ∪{PC(A)}
13:   }
14:}
15:return D
```

图 3　各结点的域编码算法

由定义 2 和 3 可知,对结点 $S$ 而言,它在路径 $P$ 上的编码 $PC(S)$ 的每一位依次对应于控制流从入口结点沿路径 $P$ 到达 $S$ 时每一个分支条件的值,0 表示分支条件为假,1 表示分支条件为真;而它的域编码 $DC(S)$ 则记录了控制流从入口结点到达 $S$ 的每一条路径上的分支条件的取值. 特别的, $DC(S) = \{nil\}$ 表示 $S$ 的谓词始终为真. 我们能够从结点 $S$ 的域编码 $DC(S)$ 中获得其谓词 $PRE_S$ 的全部定义信息. 计算结点的路径编码和域编码的工作在条件转换的同时进行,一旦完成,就能够在编译器后端的优化过程中利用域编码进行谓词分析. 图 3 给出了计算各结点域编码的算法. 如果在后端优化过程中一个结点被复制(tail-duplication)或被分割为若干个部分(split),也可根据此算法获得新结点的域编码. 而且,无论采用怎样的谓词代码生成方法,只要能够获得超块对应的控制流图,就可以用该算法得到对应的域编码. 表 1 列出了图 1(a) 中各结点的域编码.

表 1　图 1 中各结点的域编码

| 结点 | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ |
|---|---|---|---|---|---|---|---|
| 域编码 | {nil} | {0} | {1} | {11} | {10} | {0,11} | {0,11,10} |

接下来,需要定义一些路径/域编码之间的相互关系. 首先定义函数 length(str),它返回串 $str$ 的长度.

**定义 4**　两个路径编码 $PC_1$ 和 $PC_2$, $L = \min(\text{length}(PC_1), \text{length}(PC_2))$, $PC_1[i] = PC_2[i]\ (0 \le i \le L-1)$,若 $L = \text{length}(PC_1)$,则称 $PC_1$ 包含于 $PC_2$,记作 $PC_1 \subseteq PC_2$;否则称 $PC_1$ 包含于 $PC_2$,记作 $PC_1 \supseteq PC_2$.

**定义 5**　两个域编码 $DC_1$ 和 $DC_2$,若 $\forall PC_1 \in DC_1$, $\exists PC_2 \in DC_2$,使得 $PC_1 \subseteq PC_2$ 成立,则称 $DC_1$ 包含于 $DC_2$,记作 $DC_1 \subseteq DC_2$.

**定义 6**　两个域编码 $DC_1$ 和 $DC_2$ 的交集 $DC_1 \cap DC_2 = \{PC_1 \mid PC_1 \in DC_1, \exists PC_2 \in DC_2, PC_1 \subseteq PC_2\} \cup \{PC_2 \mid PC_2 \in DC_2, \exists PC_1 \in DC_1, PC_2 \in PC_1\}$.

## 2.2　基本性质

本节讨论路径/域编码的基本性质. 按照定义 2 和 3 得到的路径编码和域编码具有以下性质:

**性质 1**　对于路径 $P$ 上的两个结点 $S_1$ 和 $S_2$, $PC(S_1) = PC(S_2) \Leftrightarrow S_1 = S_2$.

性质 1 说明每个结点 $S$ 在一条路径 $P$ 上的编码是唯一的. 可以直接由定义 2 证明,此处略去证明过程.

**性质 2**　两个结点 $S_1$ 和 $S_2$,其谓词分别为 $PRE_1$ 和 $PRE_2$,运行时 $PRE_1$ 和 $PRE_2$ 可以同时为 true 当且仅当 $DC(S_1) \cap DC(S_2)$ 不是空集.

**证明**　(1)如果运行时 $PRE_1 = PRE_2 = \text{true}$,则 $S_1$ 和 $S_2$ 都是运行时正在执行的路径 $P$ 上的结点. 假设 $S_1$ 和 $S_2$ 在 $P$ 上的编码分别为 $PC(S_1)$ 和 $PC(S_2)$,显然要么 $PC(S_1) \supseteq PC(S_2)$,要么 $PC(S_1) \subseteq PC(S_2)$. 根据定义 3, $PC(S_1) \in DC(S_1)$, $PC(S_2) \in DC(S_2)$,所以 $DC(S_1) \cap DC(S_2)$ 不是空集. (2)如果 $DC(S_1) \cap DC(S_2)$ 不是空集,则存在 $PC(S_1) \in DC(S_1)$, $PC(S_2) \in DC(S_2)$, $PC(S_1) \subseteq PC(S_2)$ 或 $PC(S_1) \supseteq PC(S_2)$ 成立,不妨设 $PC(S_1) \subseteq PC(S_2)$,对应的路径为 $P$. 如果运行时路径 $P$ 被执行, $PRE_1 = PRE_2 = \text{true}$,即可以同时为真.

**性质 3**　两个结点 $S_1$ 和 $S_2$,其谓词分别为 $PRE_1$ 和 $PRE_2$,如果运行时 $PRE_2 \Rightarrow PRE_1$,则 $DC(S_1) \subseteq DC(S_2)$.

**证明**　假设 $DC(S_1) \subseteq DC(S_2)$ 不成立,即 $\exists PC(S_1) \in DC(S_1)$, $\forall PC(S_2) \in DC(S_2)$,但 $PC(S_1) \subseteq PC(S_2)$ 不成立. 这有两种可能: $\text{length}(PC(S_2)) < \text{length}(PC(S_1))$,或 $\text{length}(PC(S_1)) \le \text{length}(PC(S_2))$ 但 $\exists 1 \le j \le L$, $PC(S_2)[j] \ne PC(S_1)[j]$. 若 $\text{length}(PC(S_2)) < \text{length}(PC(S_1))$,显然当 $PRE_1 = \text{true}$ 时 $PRE_2 = \text{true}$ 不一定成立,因为 $S_2$ 并非一定位于 $S_1$ 所在的路径上. 至于第二种情况,在 $PRE_2$ 为真时 $PRE_1$ 必定为假. 假设不成立, $DC(S_2) \subseteq DC(S_1)$.

**性质 4**　两个结点 $S_1$ 和 $S_2$,其谓词分别为 $PRE_1$ 和 $PRE_2$,如果 $DC(S_1) \subseteq DC(S_2)$ 且 $|DC(S_1)| \ge |DC(S_2)|$,那么运行时 $PRE_2 \Rightarrow PRE_1$. 这里 $|DC|$ 表示集合 $DC$ 中元素的个数.

**证明**　由于 $DC(S_1) \subseteq DC(S_2)$,则 $\forall PC(S_1) \in DC(S_1)$, $PC(S_2) \in DC(S_2)$, $PC(S_1) \subseteq PC(S_2)$. 因为 $|DC(S_1)| \ge |DC(S_2)|$,根据抽屉原则, $\forall PC(S_2) \in DC(S_2)$, $\exists PC(S_1) \in DC(S_1)$, $PC(S_1) \subseteq PC(S_2)$. 如果运行时 $PRE_2 = \text{true}$,说明运行时控制流从入口结点到达 $S_2$, $S_1$ 也必定在此路径上,否则就与上面的结果矛盾,所以 $PRE_1 = \text{true}$.

性质 2 说明如何根据域编码判断谓词之间是否存在互斥关系,性质 3 和 4 说明如何根据域编码判断两谓词是否存在蕴含关系,3.1 节将讨论这些性质的应用. 按照定义 3,如果一个结点位于多条路径之上,它的域编码也应有多个元素,但实际情况并非如此. 以图 1(a) 中结点 $S_7$ 为例,它的域编码 $DC$

$(S_7)=\{1,01,00\}$,其中三个路径编码分别对应三个布尔变量:$(a<b)$,$!(a<b)\wedge(c!=d)$和$(a<b)\wedge(c!=d)$,从下面的等式可以看出,无论变量 $a$、$b$、$c$、$d$ 的值如何,$S_7$ 的谓词总为 true,即 $S_7$ 总会被执行:

$$PRE_7=(a<b)\vee(!(a<b)\wedge(c!=d))\vee((a<b)\wedge(c!=d))=\text{true}$$

因此 $DC(S_7)=\{nil\}$,其谓词恒为真.这说明,满足一定条件的域编码可以被简化,如下面的定理 1 所示.

**定理 1** 域编码的简化.结点 $S$ 的域编码为 $DC(S)$,路经编码 $PC_1$、$PC_2\in DC(S)$,如果以下条件成立:(1) $\text{length}(PC_1)=\text{length}(PC_2)$;(2) $PC_1[\text{length}(PC_1)]=PC_2[\text{length}(PC_2)]$;(3) $PC_1[i]=PC_2[i]\ (1\le i<\text{length}(PC_1))$.

令 $PC'=\text{copy}(PC_1,1,\text{length}(PC_1)-1)$

则 $DC(S)\Leftarrow$等价于 $DC'(S)=DC(S)-\{PC_1,PC_2\}+\{PC'\}$.这里函数 $\text{copy}(str,pos,len)$ 返回串 $str$ 长度为 $len$,起点为 $pos$ 的子串.定理 1 可以根据定义 2 证明,此处省略证明过程.通过合并域编码能够更直接地表示谓词定义信息,降低谓词关系分析的复杂度,提高运算效率.

## 3 谓词分析系统

### 3.1 局部谓词分析

本节我们主要讨论如何在超块范围内分析谓词关系,我们称之为局部谓词分析.对于非谓词代码,控制流图指明了每条指令的读取条件(fetch condition),在不考虑猜测执行的情况下,指令读入就一定会执行,读取条件等价于执行条件(execution condition).为保证指令调度或数据流分析结果的正确,必须考虑指令在控制流图中与其他指令的相对位置[11].在谓词执行模型中,只有当指令被读出且其谓词为真时才能执行,此时控制流图仅仅指明了指令的读取条件,只有读取条件和谓词同时为真时指令才能执行.谓词指令优化必须同时考虑指令的读取条件以及其谓词的值.在谓词执行模型中我们必须引入谓词关系,并根据谓词关系重新定义指令关系.例如,在非谓词执行模型中,指令执行蕴含关系 $fdom$(fetch dominance)说明当指令 $I_1$ 执行时指令 $I_2$ 是否一定会执行,可定义为:

$I_1\ fdom\ I_2$ *iff* 每条从入口节点到 $I_2$ 的路径都包含 $I_1$

而在谓词执行模型中此定义应修改为 $edom$(execute dominance):$I_1\ edom\ I_2$ *iff* $I_1\ fdom\ I_2$ 且 $PRE_{I1}\supseteq PRE_{I2}$

对于其他指令关系,也可以采用类似方法引入谓词关系后重新定义,本文不再赘述.与非谓词执行模型相比,主要应引入以下两种谓词关系:(1) 互斥关系(disjoint);(2) 蕴含关系(imply)由 2.2 节的几个性质可知,分析上述谓词关系能够通过分析域编码实现.

### 3.2 谓词关系查询接口

根据 3.1 节中谓词关系分析的需要,我们实现了以下接口函数,图 4 给出了这些函数的伪代码.

(1) $\text{in}(PC_1,PC_2)$:$PC_1$ 和 $PC_2$ 都是路径编码,该函数判断两个路径编码之间的关系,若 $PC_1\subseteq PC_2$ 则返回 true,否则返回 false;(2) $\text{in}(PC,DC)$:$PC$ 是路径编码而 $DC$ 为域编码,该函数判断路径编码与域编码之间的关系,若 $\exists PC_2\in DC$ 且 $PC\subseteq PC_2$,则返回 true,否则返回 false;(3) is_disjoint$(DC_1,DC_2)$:$DC_1$ 和 $DC_2$ 都是域编码,判断两个域编码的交集 $DC_1\cap DC_2$ 是否为空集.当 $DC_1\cap DC_2$ 是空集时返回 true,否则返回 false;(4) subset$(DC_1,DC_2)$:$DC_1$ 和 $DC_2$ 都是域编码,判断 $DC_1$ 是否是 $DC_2$ 的子集,是则返回 true,否则返回 false;(5) equivalent$(DC_1,DC_2)$:$DC_1$ 和 $DC_2$ 都是域编码,判断 $DC_1$ 与 $DC_2$ 是否等价,是则返回 true,否则返回 false.

### 3.3 全局谓词关系分析

为了提高谓词分析的效果,必须将分析范围从超块扩大至全局(函数)范围,为此需要扩展域编码和路径编码的定义,在全局范围表示谓词定义信息.经过条件转换,函数被划分为若干个超块,我们可以用一个二元组 $<H,DC(S)>$ 表示结点 $S$ 在超块 $H$ 中的域编码.由于经过条件转换后,超块在全局是唯一的,因此通过该二元组定义的全局域编码也是唯一的.

利用上面的二元组,我们可以实现全局谓词关系分析.分析过程应修改为:首先判断结点是否在同一个超块内,然后利用 3.2 节介绍的查询接口函数进行谓词关系分析.将谓词定义扩展到全局范围能够更好地支持全局指令调度和寄存器分配等优化措施.

```
is_disjoint(DC1, DC2)
1: for each element PC of DC1
2:     if In(PC, DC2) then
3:         return true
4:     end if
5: return false

in(PC, DC)
1: for each element PCi of DC
2:     if in(PC, PCi) then
3:         return true
4:     end if
5: return false

equivalent(DC1, DC2)
1: if(DC1 ⊆ DC2) and
   (DC1 ⊇ DC2) then
2:     return true
3: else
4:     return false
5: end if

in(PC1, PC2)
1: L = min(length(PC1), length(PC2))
2: for i = 0 to L - 1 do
3:     if(PC1[i] = PC2[i]) then
4:         return fatse
5:     end if
6: end for
7: return true

subset(DC1, DC2)
1: for each element PC of DC1
2:     if not in(PC, DC2) then
3:         return false
4:     end if
5: return true
```

图 4 局部谓词关系分析算法

## 4 性能分析

### 4.1 复杂度分析

从图 4 可以看出,$\text{in}(PC_1,PC_2)$ 是谓词查询结口中最基本的函数,实际上它可以用下面的表达式实现:

$$\text{in}(PC_1,PC_2)=((2^L-1)\ \&\ (PC_1\oplus PC_2)==0)$$

这里 $L=\min(\text{length}(PC_1),\text{length}(PC_2))$,&为按位与操作,$\oplus$ 为按位异或操作,==为布尔操作.显然该函数的时间复杂度为 $O(1)$.由于控制流图中每个结点最多只有 2 个后继,因此对一个含有 $N$ 个结点的 CFG 而言,它最多有 $2^{\log_2(N+1)-1}<N$ 条路径.表 2 列出了几个接口函数的平均时间复杂度,而现有

的算法分析互斥、蕴涵等关系时的复杂度一般为 $O(|N|^2)$[4,7~9]. 保存所有域编码所需的空间复杂度最多为 $O(|N|^2)$. 这里 $|N|$ 表示集合 $N$ 中元素的个数.

表 2　接口函数的时间复杂度

| 函数名 | in($PC_1$, $PC_2$) | in($PC$, $DC$) | is_disjoint | Subset | include |
|---|---|---|---|---|---|
| 复杂度 | $O(1)$ | $O(1)$ | $O(\|N\|)$ | $O(\|N\|)$ | $O(\|N\|)$ |

### 4.2　模拟测试结果

为了评价该谓词分析系统的效能,我们针对一组基准程序进行模拟测试,并根据实际运行时间进行评价. 这组基准程序由 SPEC INT95 中的 130.li,129.compress,124.m88ksim,以及 UNIX 环境下的 cccp,lex 和 wc 组成,均由 IMPACT 2.36 编译并优化. 测试在主频为 1.4GHz,RAM 为 256MB 的硬件环境下进行,操作系统为 Red Hat 8.0. 为了确定谓词关系查询接口的性能,对每个超块中的每一对谓词,我们都测试它们之间的 is_disjoint,subset 和 equivalent 关系. 一共进行了 837,315 项查询操作,在 2.17 秒内完成. 正是由于域编码的良好特性,才能获得如此高的响应速度. 大约 6.3%的查询操作将直接影响编译优化的性能,对于没有进行谓词分析的编译系统,如果没有进行这些操作,编译优化时只能获得保守的结果.

图 5 主要针对性能进行比较,其中白色表示使用谓词分析系统进行谓词关系分析及优化后得到的性能提升(平均 20.7%),而黑色表示使用 IMPACT 系统采用基于超块的编译优化技术获得的加速比(平均 17.6%). 可以看出,除了 wc 以外,其余基准程序都得到了较大的性能提高. 这是因为在 wc 中谓词指令相对比较少,因此无法体现出基于谓词的优化的效果. 这说明采用本文介绍的方法能够获得更高质量的代码.

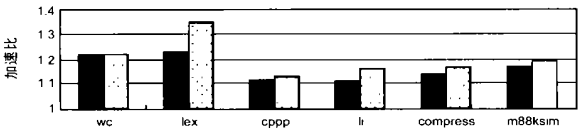



图 5　基于路径信息的谓词分析技术能够提高代码质量

该方法对编译器的性能影响也很小,这可以从表 3 中看出. 与图 5 相同,表 3 分别列出了使用 IMPACT 超块编译技术以及使用本谓词分析系统的编译时间,大部分程序的编译时间有所减少. 对于有些程序,编译时间增加的一个主要原因是程序中分支结构比较复杂,但与最后获得的性能加速比相比,这是完全可以容忍的. 结合图 5 与表 3 我们可以看出,该方法能够在提高代码质量的基础上有效提高系统性能.

表 3　基于路径信息的谓词分析对编译时间的影响很小

| 基准程序 | wc | lex | cppp | li | compress | m88ksim |
|---|---|---|---|---|---|---|
| IMPACT | 0.64 | 0.91 | 1.04 | 4.58 | 3.34 | 3.67 |
| 基于路径的谓词分析 | 0.77 | 0.89 | 1.07 | 4.03 | 3.66 | 3.40 |

## 5　结束语

本文主要研究了如何高效精确地实现谓词关系分析. 文中提出了一个基于路径信息的谓词表示方法,将谓词定义信息转换为路经编码和域编码. 由于保存域编码需要的空间开销很小,因此可以避免现有工作中重构谓词定义信息的过程,从而降低了操作的复杂度. 以这种表示方法为基础,我们设计并实现了一个全局谓词关系分析系统并给出了系统接口函数的实现算法,该系统能够以线性时间复杂度完成谓词关系分析. 模拟结果表明,该系统能够高效精确地实现全局谓词关系分析,能够提高目标代码的质量.

**参考文献**:


[1] P Y Hsu, E S Davidson. Highly concurrent scalar processing [A]. Proc. of the 29th Annual Int'l Symp. on Microarchitecture [C]. Tokyo: IEEE Computer Society Press, 1996. 386 - 395.

[2] U Kathail, M Schlansker, B Rau. HPL PlayDoh Architecture Specification: Version 1.0 [R]. Hewlett-Packard Laboratories Technical Report, HPL-93-80, Feb. 1993.

[3] J R Allen, K Kennedy, C Portfield, J Warren. Conversion of control dependence to data dependence [A]. In Conf. Record of the 10th Annual ACM Symp. on Principles of ProgrammingLanguages [C]. Austin: ACM Press, 1983. 177 - 189.

[4] S A Mahlke, D C Lin, W Y Chen, R E Hank, R A Bringman. Effective compiler support for predicated execution using the hyperblock [A]. Proc. of the 25th Annual International Symposium on Microarchitecture [C]. Portland: IEEE Computer Society Press, 1992. 45 - 54.

[5] G S Tyson. The effects of predicated execution on branch prediction [A]. Proc. of the 27th Annual International Symposium on Microarchitecture [C]. New York: ACM Press, 1994. 196 - 206.

[6] S A Mahlke, R E Hank, R A Bringman, J C Gyllenhaal, D M Gallagher, W Hwu. Characterizing the impact of predicated execution on branch predication [A]. Proc. of the 27th Annual International Symposium on Microarchitecture [C]. San Jose: ACM Press, 1994. 217 - 227.

[7] Alexander E Eichenberger, E S Davidson. Register allocation for predicated code [A]. Proc. of the 28th Annual Int'l Symp. on Microarchitecture [C]. Ann Arbor: IEEE Computer Society Press, 1995. 180 - 191.

[8] Richard Johnson, Michael Schlansker. Analysis techniques for predicated code [A]. Proc. of the 29th Annual Int'l Symp. on Microarchitecture [C]. Paris: IEEE Computer Society Press, 1996. 100 - 113.

[9] J W Sias, Wen-mei W Hsu, D I. August. Accurate and efficient predicate analysis with binary decision diagrams [A]. Proc. of the 33rd ACM/IEEE International Symposium on Microarchitecture [C]. Monterey: IEEE Computer Society Press, 2000. 112 - 123.

[10] A Aho, R Sethi, J Ullman. Compilers: Principles, Techniques, and Tools [M]. Addisom-Wesley, Reading, MA, second edition, 1986.


**作者简介**:

沈　立　男,1976 年生于陕西,主要从事微处理器系统结构,编译优化技术等方面的研究.