软件流水中隐式控制流恢复技术*)

汪 森[1,2] 赵荣彩[1] 蔡国明[3] 丁志芳[2]
(解放军信息工程大学信息工程学院 郑州 450002)[1] (解放军信息工程大学理学院 郑州 450001)[2]
(解放军信息工程大学电子技术学院 郑州 450004)[3]

摘 要 具有条件分支的循环通过 IF 转换将显式的控制流转换为隐式的控制流,从而为指令调度提供进一步的机会。但它往往将程序的代码进行深度重构,增加了程序的理解和代码重建工作的复杂性。提出了一种软件流水循环中的隐式控制流恢复技术,用于重构软件流水循环中的条件分支,提高软件逆向工程中生成的目标代码的质量。

关键词 软件流水,IF 转换,谓词执行,寄存器旋转

Recovery of Implicit Control Flow in Software Pipelined Loops

WANG Miao[1,2] ZHAO Rong-cai[1] CAI Guo-ming[3] DING Zhi-fang[2]
(Institute of Information Engineering, Information Engineering University, Zhengzhou 450002, China)[1]
(Institute of Science, Information Engineering University, Zhengzhou 450001, China)[2]
(Institute of Electronic Technology, Information Engineering University, Zhengzhou 450004, China)[3]

Abstract If-conversion with predicated execution has been proposed to convert explicit conditional branches in loops into implicit control flow, which can enlarge the chance of instruction schedule. However it can profoundly restructure the low level code of programs and complicate the task of understanding and re-engineering from an optimized executable. This paper describes a technique to recover implicit control flow in software pipelined loops and thereby improve the quality and efficiency of IA-64 binary translation. This technique has been implemented in our static binary translator and is proved to be valid with the experiments.

Keywords Software pipelining, If-conversion, Predicated execution, Register rotation

1 引言

分支指令是制约微处理器指令并行能力的主要障碍之一。尤其是包含于循环中的条件分支,给发掘循环中的指令级并行性造成了困难。软件流水[1]是一种开发循环程序指令级并行性的技术。对于具有条件分支的循环,应用软件流水优化时,一种常用的方法是通过 IF 转换[2]隐藏显式的控制流,把对应的控制相关转变成数据相关,从而扩大基本块的大小,给软件流水优化提供进一步的机会和便利。

以 IA-64 为代表的 EPIC(Explicity Parallel Instruction Computing)体系结构使用了谓词执行技术(Predicate execution)[3],通过为每条指令增加一个谓词寄存器参数来尽可能多地避免由条件语句导致的跳转。虽然谓词执行可以很好地挖掘底层处理器的先进特征,但是也给分析和翻译具有谓词执行特性代码的工作提出了挑战。谓词执行需要对程序的低级代码进行深度重构,隐藏了源程序中的控制流,使得代码分析程序难以从优化后的代码中恢复源程序的逻辑,从而使得静态分析和修改可执行文件的工作变得异常复杂,例如逆向工程系统、静态二进制翻译和连接优化程序等。尤其是软件流水优化后的循环代码,恢复其中的隐式控制流更加困难。

目前,从带谓词的代码中重构隐式控制流已有一些相关的研究。Decker 等人在 PROPAN 系统中提出了一种基于静态语义推理的控制流重构算法[4]。Snavely 等人提出了一种智能反谓词算法[5],该算法在谓词分析的基础上,向控制流图中添加新的基本块和边,替换带谓词的指令完成指令的谓词消除,恢复原始程序控制流。但这些研究都没有考虑软件流水循环中的隐式控制流恢复。反向 IF 转换技术[6]用于将 IF 转换后的代码转换为控制流图表示,它应用于增强型模调度算法[7]中。增强型模调度算法首先使用 IF 转换消除循环中的条件分支,在此基础上进行模调度,调度后再使用反向 IF 转换重新产生显式的控制结构。但是这种反向 IF 转换技术需要依据前面 IF 转换阶段留下的标记。而这样的标记在一般的可执行文件中是没有的。

本文提出了软件流水循环内的隐式控制流恢复技术,在保证不影响程序翻译正确性的前提下,可以有效地识别软件流水循环中的条件分支结构,提高软件逆向工程生成代码的质量。

2 相关知识概要

2.1 谓词执行

IA-64 架构提供了 64 个 1 位的谓词寄存器。大部分指令都可以通过指令前的谓词来实现判定执行的功能。如果该谓词寄存器值为真,则指令执行;否则,不执行。

谓词寄存器可通过谓词定义指令设置,共有三大类谓词定义指令,分别是 normal, unconditional 以及 parallel。normal 类型将两个数据操作数的比较结果及其补分别赋值给相应的

*)基金项目:国防重点科研项目资助。汪 森 博士研究生,研究方向为计算机软件与理论;赵荣彩 教授,博士生导师,研究方向为计算机软件与理论;蔡国明 博士研究生,研究方向为计算机软件与理论;丁志芳 副教授,研究方向为计算机软件与理论。

谓词寄存器。unconditional 类型同 normal 比较类似，差别在于它在进行数据比较并设置结果之前将两个谓词寄存器操作数清零，此外，即便这条比较指令由于其本身的谓词为假而没有执行，也对两个谓词寄存器操作数清零。parallel 类型用于复合条件比较。

谓词结合指令调度可以减少明显的分支指令以及增加相当数量的指令并行性。例如，C 语句

```
if(cond)
    inst1;
else
    inst2;
```

经过编译器优化生成下面带谓词形式的机器代码：

```
cmp. eq p6,p7= r36,r38;;
(p6) instrl
(p7) instr2
```

其中谓词寄存器 p6，p7 的值根据寄存器 r36 和 r38 比较结果设置为互补。经过优化后的机器代码避免了两个分支：一个跳转到 else 基本块，另一个跳转到 else 基本块以外。

2.2 软件流水

软件流水是将一个循环中的指令实现模块调度，像硬件中的指令流水一样，软件流水将属于同一循环的不同迭代的指令序列并行执行。

IA-64 引入了两个新特性来支持软件流水：旋转寄存器和特殊的软件流水分支指令。旋转寄存器机制使得编译器不必显式地进行寄存器重命名。在 IA-64 中，指令中指定的是逻辑寄存器编号，其实际要访问的物理寄存器编号等于逻辑寄存器编号加上寄存器重命名基寄存器(rrb)的值。rrb 的值是由 IA-64 中的软件流水分支指令来控制的。每执行一条软件流水分支指令，rrb 减 1，使得循环体 I 中逻辑寄存器 X 的内容，在循环体 I+1 中移到了逻辑寄存器 X+1 中。特殊的软件流水分支指令(包括 br. ctop，br. cexit，br. wtop，br. wexit)可以准确地预测循环结束信息，加速软件流水循环的执行。

3 软件流水中的隐式控制流恢复技术

3.1 问题描述

当循环中含有条件分支语句，应用软件流水优化时，一种常用的方法是通过 IF 转换隐藏显式的控制流，把对应的控制相关转变成数据相关，从而给软件流水优化提供进一步的机会。

例如，如下的一个 c 语言循环程序段

```
for (i = 0; i < alphaSize; i++)
    if(freq[i] >=0)
        weight[i+1] =freq[i] << 8;
    else
weight[i+1] =(-freq[i])<<8;
```

经过 ICC8.0 编译器 2 级优化后可得到如下的软件流水代码：

```
            mov. i ar. ec=3
            mov pr. rot=0x10000;;
            mov. i ar. lc=r9
L:          I1:(p20) sub r40=r0,r38
            I2:(p16) ld4 r37=[r2],4
            I3:(p22) shl r39=r38,8
            I4:(p21) st4 [r35]=r36;;
            I5:(p16) cmp4. lt. unc p19,p21=r37,r0
            I6:(p20) shl r35=r40,8
            I7:(p22) st4 [r34]=r39
            I8:(p16) adds r32=4,r33
            br. ctop L
```

虽然经过编译优化后得到的代码使程序性能得到明显的提升，但对程序理解和逆向工程来说造成了很大的困难。首先，源循环中的条件分支通过 IF 转换被隐藏为隐式的控制流，因此软件流水循环中没有显式的分支指令。如例中所示，谓词定义语句 I5 定义了两个谓词 p19，p21，分别用来控制 if 语句两个分支的执行。其次，由于寄存器发生了旋转，定义的谓词可能和实际用来限定分支执行的谓词在逻辑寄存器号上是不一样的。例如定义的谓词 p19，p21 是在软件流水的第一个阶段定义的。在第二阶段，由于发生了一次旋转，实际用来限定分支执行的限定谓词是 p20，p22。同理，在第三阶段，限定谓词就变成了 p21，p22。另外，由于指令调度改变了指令在源程序中的顺序和位置，属于同一个分支的指令可能不是在一起的，也可能不处于同一个流水阶段中。指令 I1，I3，I6 的限定谓词分别是 p20，p21，p20。但它们实际上都是属于一个分支(满足 r37<0 时)的指令。

从上面的分析可以看出：软件流水和谓词执行的结合使用，使得低级代码晦涩难懂，并且增加了程序的理解、分析和逆向工程的难度。因此，在对优化后的可执行文件进行逆向分析时，需要恢复软件流水循环中的隐式控制流。

3.2 恢复算法

我们的恢复算法包括 4 个步骤：首先计算软件流水中每条指令的阶段号，然后根据阶段号将指令重新进行排序，第三步恢复旋转寄存器号，最后利用智能反谓词算法恢复原始的程序控制流。

3.2.1 计算指令的阶段号

我们将软件流水循环中的谓词按照作用分为两类：阶段谓词和比较谓词。阶段谓词用于控制软件流水阶段的执行。对于计数型循环来说，规定 p16 用来控制软件流水第 1 阶段的执行，p17 用来控制第 2 个阶段的执行，等等。在软件流水循环前使用 mov pr. rot 指令给阶段谓词赋初值。如果指令前的谓词是阶段谓词，可直接根据它判断出指令对应的阶段号。

比较谓词是由循环体内的谓词定义指令定义的，用于控制循环中分支的执行。如果指令前的谓词是比较谓词，它的阶段号取决于谓词定义指令所处的阶段和寄存器旋转的次数。

计算指令阶段号的算法如下所示，其中 SP 表示阶段谓词集合，CP 表示比较谓词集合。

输入：软件流水循环体指令集合 I，阶段数 SC

输出：循环中每条指令的阶段号

算法：

```
初始化 I 中每条指令的阶段号为∞
初始化 SP={16},CP=Ø
for curstage=1 to SC do
  for 指令 i∈I do
    如果指令 i 的阶段号大于 curstage
    If i 的限定谓词属于 SP 或 CP
    {设置 i 的阶段号为 curstage;同量如果 i 是谓词定义指令在 CP 中增加定义的谓词}
  else{
```

如果 i 是一条无条件比较指令并且 i 定义的谓词已在 CP 中，

end for

curstage++;

SP、CP 中的谓词加 1

end for

3.2.2 指令重排序

由于对于同一个迭代来说，总是先执行第一个阶段的指令，再执行第二个阶段的指令，然后依次，因此按照阶段号将软件流水循环中的指令进行重排序。对于同一个阶段的指令，维持指令原来的先后顺序。

3.2.3 恢复旋转寄存器号

在正式描述这个问题前，我们定义旋转寄存器的等价关系。

定义 1 两个逻辑寄存器编号不同的旋转寄存器 r1 和 r2，如果其实际访问的物理寄存器编号是一致的，则称这两个旋转寄存器是等价的，r1 和 r2 互为等价关系。

在本算法中，我们的目的是要消除旋转寄存器的影响。我们的思路是保留第一阶段的寄存器索引不变，将其它阶段的寄存器号用与它等价的第一个阶段的寄存器来替换。例如：I1 指令中的旋转寄存器 p20，r40，r38(第 2 阶段)分别用与它等价的 p19，r39，r37(第 1 阶段)来替换。

3.2.4 反谓词

通过上面 3 个步骤消除了软件流水优化的影响后，我们利用智能反谓词算法恢复条件分支。通过数据流分析确定指令中谓词寄存器之间的关系，在控制流图中添加新的基本块和边，替换带谓词的指令，完成指令的谓词消除，恢复原始程序控制流。具体算法如下：

输入：软件流水循环基本块 B

输出：B 恢复隐式控制流后对应的控制流图 GB

算法：

通过谓词分析确定 B 中互补关系谓词集合、支配关系谓词集合确定谓词支配链

根据谓词支配链之间的关系识别谓词组

对每一个谓词组，将谓词组中的每一条指令都拆开放入到新的基本块，然后调整基本块之间的边。

4 实验结果

本文提出的隐式控制流恢复技术在静态二进制翻译系统 I2A 中进行了实现。以下是示例程序翻译后的低级 C 语言代码片段。

```
    for(i=lc;i>=0;i--)
{
  *(unsigned int64 *)&r37=(unsigned int64)(*((unsignedint32
*)(*(unsigned int64 *)&r2)));
  r2=(r2)+(4);
  if(r37<0)
  {
  r39=0-r37;
  r34=r39<<8;
  *((int32 *)(*(unsigned int64 *)&r33))=(int32)(r34);
  }
  else
  {
  r38=437<<8;
  *((int32 *)(*(unsigned int64 *)&r33))=(int32)(r38);
}
  r33=r33+4;
}
```

可以看出翻译后的低级 c 代码已经正确地恢复出了循环中的条件分支。翻译后得到的低级 c 代码在 Alpha 上经编译后运行结果同源程序在 IA-64 上运行结果一致。

目前 I2A 系统已经成功翻译了 CEXAMPLES，SPEC CPU2000，NAS，SUPTERTEST 测试集中的 1000 多个测试用例。表 1 列出了部分实验结果。测试用例来自 SPECint-2000 基准测试集中的 8 个例子，用编译器 ICC8.0，优化选项-O2 编译后得到的可执行文件。表 1 中列出了每个例子中出现的计数型软件流水循环个数以及其中恢复出隐式控制流的循环个数。

表 1 实验结果

测试用例	计数型循环个数	恢复出隐式控制流的循环个数
256. bzip2	18	7
164. gzip	11	1
181. mcf	3	2
197. parser	9	5
254. gap	81	21
255. vortex	3	0
186. crafty	28	12
300. twolf	103	51

结束语 本文在对 IA-64 谓词执行机制和软件流水代码机制进行分析的基础上，提出了一种恢复软件流水循环代码中隐式控制流的算法。该算法在二进制翻译框架中进行了实现和检测，算法的正确性得到了充分的验证。针对优化后的可执行文件的逆向工程仍存在许多难题，这将是我们下一步研究的重点。

参 考 文 献

[1] Allen V H, Jones R B, Lee R M, et al. Software pipelining. ACM Computing Surveys, 1995, 27(3): 367-432

[2] Allen J R, Kennedy K, Porterfield C, et al. Conversion of control dependence to data dependence // POPL '83: Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York, NY, USA, ACM Press, January 1983: 177-189

[3] Intel Corporation. Intel Itanium™ Architecture Software Developer's Manual. Volume 1: Application Architecture. Intel Corporation, 2001

[4] Decker B, Kastner D. Reconstructing control flow from predicated assembly code // Software and Compilers for Embedded Systems, 7th International Workshop (SCOPES'03). volume 2826 of Lecture Notes in Computer Science. Springer, September 2003: 81-100

[5] Snavely N, Debray S, Andrews G. Unscheduling, unpredication, unspeculation: Reverse engineering Itanium executables // (WCRE '03): Proceedings of the 10th Working Conference on Reverse Engineering. Washington, DC, USA, IEEE Computer Society, 2003: 4-13

[6] Warter N J, Mahlke S A, Hwu W M W, et al. Reverse if-conversion // (PLDI '93): Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and implementation. New York, NY, USA, ACM Press, 1993: 290-299

[7] Warter N J, Haab G E, Subramanian K, et al. Enhanced modulo scheduling for loops with conditional branches // MICRO 25: Proceedings of the 25th Annual International Symposium on Microarchitecture. Los Alamitos, CA, USA, ACM and IEEE, December 1992: 170-179