计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20130018
51(10):2336-2347,2014

二进制翻译中标志位的模式化翻译方法

王文文[1,2] 武成岗[1] 白童心[1] 王振江[1] 远 翔[1,2] 崔慧敏[1]
[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](中国科学院大学 北京 100049)
(wangwenwen@ict.ac.cn)

A Pattern Translation Method for Flags in Binary Translation

Wang Wenwen[1,2], Wu Chenggang[1], Bai Tongxin[1], Wang Zhenjiang[1], Yuan Xiang[1,2], and Cui Huimin[1]
[1](*State Key Laboratory of Computer Architecture*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Binary translation is an important way for software migration between different hardware platforms. For binary translation systems, how to simulate the flag register on target platform that does not have the function of the flag register, is a key factor affecting system performance. Previous flag analysis techniques delete the unnecessary flag computing via the flow analysis of flag definition and usage during the process of flag translation. However, for the necessary flag computing, these techniques still have to translate for simulation, which can lead to the expanse of generated native code and the loss of system performance. A pattern translation method is presented for necessary flag computing in binary translation system. Based on previous flag analysis techniques, this method makes flag definition instruction and flag usage instruction a flag pattern, and chooses appropriate instruction groups on target platform to translate flag computing based on the semantics of the flag pattern. This method for flag translation can reduce the expanse of generated native code and improve the performance of binary translation system. The experiment results demonstrate that, for the programs in SPEC CINT2006, this method can reduce the amount of generated native code by 7.5% and improve the system performance by 10% on average.

Key words binarytranslation; flag register; flag translation; flag pattern; code optimization

摘 要 二进制翻译是在不同硬件平台之间实现软件迁移的重要手段.在二进制翻译系统中,如何在没有标志位寄存器的目标平台上模拟实现源平台上标志位寄存器的功能,是影响系统性能的关键.现有的标志位分析技术通过对标志位的定值引用进行数据流分析,尽可能多地消除冗余的标志位定值.但是,对于那些会被引用的标志位定值,现有的技术仍然需要进行翻译.这不仅会导致翻译生成代码的膨胀,还会影响二进制翻译系统的性能.提出了一种二进制翻译中基于模式化的标志位翻译方法.该方法在标志位分析技术基础上,通过将源平台上标志位定值指令和引用指令组合成固定的标志位模式,然后根据模式的具体语义选择目标平台上具有相同语义功能的指令组合进行翻译.这种模式化的翻译方法,不仅可以降低因翻译标志位而引入的代码膨胀,还可以提升二进制翻译系统的性能.实验结果表明,对于SPEC CINT2006中的程序,该方法不仅可以使翻译生成的代码量平均减少7.5%,还可以将程序的性能平均提升10%.

收稿日期:2013-01-04;**修回日期**:2013-04-09
基金项目:国家"八六三"高技术研究发展计划基金项目(2012AA010901);国家自然科学基金青年科学基金项目(61100011);国家自然科学基金杰出青年基金项目(60925009)

关键词 二进制翻译;标志位寄存器;标志位翻译;标志位模式;代码优化

中图法分类号 TP311

二进制翻译使用软件的方法在一种处理器平台(目标平台)上,翻译执行另一种处理器平台(源平台)的二进制代码.通过这种方式,二进制翻译可以实现不同平台之间的代码迁移,进而解决遗产代码的移植问题.从20世纪80年代开始,二进制翻译就已经被广泛地应用在计算机研究和工业领域中,常见的系统如Pin[1],Valgrind[2],FX!32[3]和IA-32 EL[4]等.

在当前计算机的处理器市场中,Intel 64/IA-32占据着桌面电脑和服务器的主流市场;而ARM则占据着嵌入式设备(如智能手机、平板电脑等)的主流市场.在这类处理器上有着丰富的应用程序.同时,软件开发商也愿意为这类市场占有率高的处理器开发应用软件.而对于另一类处理器,如MIPS,IA-64,Alpha和Loongson等,它们的市场占有率没有第1类处理器那么高.为了提升市场竞争力,它们需要有丰富的软件支撑;但是另一方面,如果没有足够的市场,那么软件开发商也不愿意为它们开发新型的软件.这两个因素相互制约,导致第1类处理器的市场占有率不高.因此,研究以第1类处理器为源平台、第2类处理器为目标平台的二进制翻译,对于提升第2类处理器的市场竞争力、繁荣处理器市场具有重要的实际意义.

在上述源平台处理器中,标志位寄存器是用于记录标志位的关键部件.指令可以根据标志位进行不同的操作,从而实现不同的程序语义,如根据标志位取值的不同确定是否进行分支的条件分支,以及根据标志位取值的不同确定是否进行置位的条件置位等.但是在上述目标平台处理器中,却没有标志位寄存器.如果要在目标平台上翻译源平台上的应用程序,就需要对标志位寄存器中的标志位进行翻译.所谓标志位的翻译就是根据源平台指令对标志位的操作语义,在目标平台上翻译生成具有相同功能的指令,通过执行这些生成的指令来实现源平台上标志位的语义功能.

指令对标志位寄存器中某个标志位的操作有定值和引用2种,并且指令可以2种操作都有,也可以2种操作都没有.定值的意思是指令会修改这个标志位:置0或置1;引用的意思是指令会根据这个标志位的内容(0或1)进行不同的操作.如果指令对某个标志位的定值没有被后续执行的指令引用,或者在后续指令引用之前又被重新定值,那么这条指令对该标志位的定值就是冗余的定值.在二进制翻译中,翻译一条指令对单个标志位的定值一般需要5条左右的指令.如果指令定值多个标志位,那么每个标志位的定值均需要进行翻译.因此,在二进制翻译中删除冗余的标志位定值可以减少标志位的翻译,降低系统的开销,提升系统的性能.为了利用这个优化机会,很多文献提出了基于数据流分析的标志位分析技术.这些分析技术通过对被翻译程序进行标志位的定值引用分析,删除那些冗余的标志位定值.典型的如FX!32系统的延迟计算方法[3]、IA32 EL的不完全数据流图分析方法[4]、中国科学院计算技术研究所的立即计算和延迟计算相结合算法[5]、数据流和延迟计算相结合算法[5]以及基于动态反馈的标志位线性分析算法[6]等.

虽然标志位分析技术可以有效删除冗余的标志位定值,但是那些会被后续指令引用的标志位定值仍然需要进行翻译.下面通过一个具体的实例来说明标志位分析技术的不足.以IA-32[7-9]平台为例,其标志位寄存器EFLAGS有7个标志位①:*CF*,*PF*,*AF*,*ZF*,*SF*,*OF*和*DF*.图1(a)是SPEC CPU2006中473.astar程序在IA-32平台上编译生成的代码片段.通过对这段代码进行标志位定值引用分析可以发现,804a3cb处的指令CMP定值的标志位(*CF*,*OF*,*SF*,*ZF*,*AF*和*PF*)仅被804a3d1处的指令JG引用.由于指令JG引用的标志位只有3位:*OF*,*SF*和*ZF*,所以标志位分析技术可以删除冗余的标志位定值(*CF*,*AF*和*PF*).以目标平台MIPS为例,图1(b)给出了经过标志位分析后在目标平台上生成的指令序列.其中源平台到目标平台的寄存器映射如下所示:

1) EFLAGS→s8;

2) EDX→at;

3) EDI→s7;

4) ESP→s4;

5) 其他寄存器均为临时寄存器.

① 这里不考虑系统标志位,因为应用程序不允许修改系统标志位.

通过观察这段代码可以发现,虽然标志位分析技术可以删除冗余的标志位定值,但是翻译那些会被引用的标志位定值仍然会产生大量的指令.实际上,对于那些目标平台有相同语义功能的源平台指令,二进制翻译基本可以实现一对一的翻译,例如翻译 804a3cd 处的指令 MOV 仅生成 70e62e00 处的 SW 指令.而对于源平台的标志位定值和引用指令,由于目标平台没有相同语义功能的指令,因此翻译它们就需要生成多条指令,如本例中翻译 2 条标志位定值和引用指令需要生成 21 条指令,翻译后的代码膨胀达 10 倍.这对于二进制翻译系统的性能是很不利的.

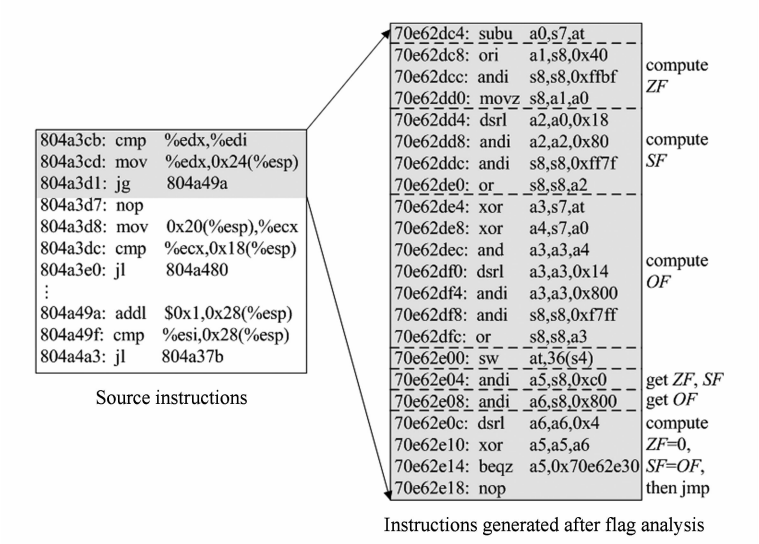


Fig. 1 The shortage of flag analysis.

图 1 标志位分析的不足

为了弥补上述标志位分析技术的不足,进一步降低因翻译标志位而引入的性能开销,本文提出了一种在二进制翻译中对标志位进行模式化翻译的方法.该方法在上述标志位分析技术的基础上,将源平台上的标志位定值指令和引用指令组合成一个标志位模式,然后根据不同模式的具体语义,在目标平台上选择具有相同语义功能的指令组合,进行标志位的翻译.这样就可以进一步删除那些需要翻译的标志位定值.这种标志位的翻译方法不仅可以降低因翻译标志位而引入的代码膨胀,还可以提升二进制翻译系统的性能.此外,本文还将该方法实现在一个已有的二进制翻译系统 DigitalBridge 中,并对 SPEC CINT2006 进行测试,实验结果表明,该方法在将翻译生成的代码量减少 7.5%的同时,还将程序的性能提升 10%.

1 标志位模式

在有标志位的处理器上,标志位一般是作为指令的条件或输入进行使用,如条件分支、条件置位等.而在没有标志位的处理器上,指令虽然无法使用标志位,但仍然可以完成相同的功能.下面以条件分支为例进行说明.

图 2(a)是一段高级语言代码,该代码对变量的值进行简单地判断,然后根据不同的判断结果执行不同的分支;图 2(b)是这段代码在有标志位的处理器 IA-32 上编译生成的指令序列;图 2(c)是这段代码在没有标志位的处理器 MIPS 上编译生成的指令序列.可以看出,在有标志位的处理器上,指令首先根据比较结果对标志位进行定值,然后再引用标志位进行条件分支;而在没有标志位的处理器上,指令可以直接根据比较结果进行条件分支.

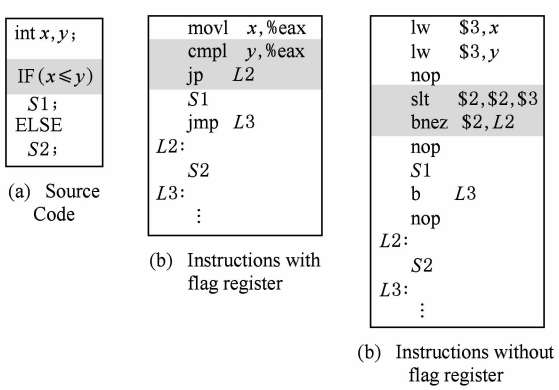


Fig. 2 Different implementations of conditional branch.

图 2 条件分支在不同处理器上的不同实现方式

如果把标志位的定值指令和引用指令看作一个模式,其中定值指令称为模式头,引用指令称为模式尾,并且模式尾引用的标志位均来自于模式头的定值,那么图 2(b)中的标志位模式就可以实现语义为"≤"的条件分支.图 2(c)中虽然没有使用任何标志位,但通过合适的指令组合也实现了相同的语义功能.标志位的模式化翻译就是在标志位分析技术的基础上,通过查找被翻译程序中的标志位模式,然后按照模式的具体语义,选择目标平台上具有相同语义功能的指令组合,实现标志位的翻译.通过这种翻译方法,那些仅由模式尾对标志位的引用而产生的标志位定值,在模式头处就不必进行翻译.这样就可以进一步减少需要翻译的标志位定值.

标志位寄存器一般含有多个标志位,而指令定值或引用的标志位可能仅仅是标志位寄存器中的部分标志位.这就导致了标志位模式的多样化结构.根据结构的不同,可以将标志位模式分为以下 3 类:单头单尾、单头多尾和多头单尾.图 3 给出了这 3 种结构的示意图,其中箭头表示标志位的定值引用关系.图 3(a)是单头单尾结构,这种标志位模式只有单个模式头和单个模式尾,模式尾引用的标志位均由模式头定值;图 3(b)是单头多尾结构,这种标志位模式含有单个模式头和多个模式尾,并且所有模式尾引用的标志位均来自于模式头的定值;图 3(c)是多头单尾结构,这种标志位模式含有多个模式头和单个模式尾,模式尾引用的标志位来自于不同模式头的定值,这是因为不同的模式头定值了不同的标志位,而模式尾引用了这些标志位.这里之所以没有多头多尾的结构,是因为这种结构可以分解使用上述 3 种结构进行表示.

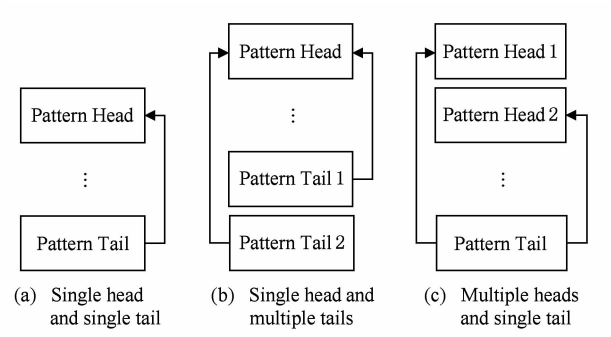


Fig. 3 Different structures of flag patterns.

图 3 标志位模式的 3 种不同结构

除了简单相互独立的布局外,不同的标志位模式在二进制代码中的布局也可能有多种,如嵌套、交叠、重用等.图 4 给出了这 3 种布局的示意图.图 4(a)是嵌套布局,这种布局表示一个标志位模式的全部均在另一个标志位模式的模式头和模式尾之间;图 4(b)是交叠布局,这种布局表示一个标志位模式覆盖的区域和另一个标志位模式覆盖的区域之间存在交叠;图 4(c)是重用布局,这种布局表示 2 个标志位模式的模式头或模式尾存在重用,如:一个标志位模式的模式尾是另一个标志位模式的模式头,即这条指令既引用标志位又定值标志位.

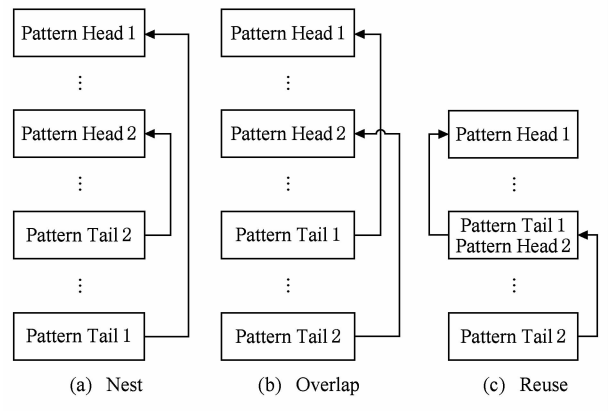


Fig. 4 Different layouts of flag patterns.

图 4 标志位模式的 3 种不同布局

虽然经过上述分析,标志位模式可能存在不同的结构和布局,但是在实际程序中,除少量的重用布局外,多头单尾的结构和嵌套、交叠的布局非常少见.这是因为:一方面,定义部分标志位的指令在实际程序中比较少见;另一方面,编译器是根据高级语言的语义进行指令的生成,而一般高级语言的语义通过简单的结构和布局即可实现,例如图 2 中的"≤"语义,所以,除非程序员特意写这样的代码,编译器一般很少会生成含有复杂结构和布局的指令序列.表 1 给出了不同结构和布局的标志位模式在 SPEC CINT2006 各输入集下的数量①.从表 1 也可以看出,大部分标志位模式均是简单布局的单头单尾和单头多尾结构.因此,在进行标志位的模式化翻译时,可以选择简单布局的单头单尾和单头多尾结构作为翻译目标.这样不仅可以充分利用标志位模式的优化机会,还可以有效降低翻译方法的复杂度.

① 虽然 SPEC CFP2006 中也存在标志位模式,但是由于浮点程序中基本块一般较大,翻译标志位对性能的影响没有整点那么明显,因此本文仅针对 SPEC CINT2006 进行测试.表 1 中同一输入集下相同标志位模式不重复计数,下同.

Table 1 The Number of Flag Patterns with Different Structures and Layouts

表 1 不同结构和布局的标志位模式数量

Applications	*test*						*train*						*ref*					
	SS	SM	MS	N	O	R	SS	SM	MS	N	O	R	SS	SM	MS	N	O	R
400. perlbench	12 037	35	0	0	0	3	12 803	37	0	0	0	0	14 788	46	0	0	0	0
401. bzip2	2 189	4	0	0	0	0	2 274	4	0	0	0	0	2 272	4	0	0	0	0
403. gcc	39 059	274	0	0	0	0	34 711	261	0	0	0	0	38 892	248	0	0	0	0
429. mcf	1 398	1	0	0	0	0	1 444	2	0	0	0	0	1 417	2	0	0	0	0
445. gobmk	13 175	47	0	0	0	0	13 914	50	0	0	0	0	14 755	51	0	0	0	0
456. hmmer	2 135	5	0	0	0	0	2 155	5	0	0	0	0	2 568	6	0	0	0	0
458. sjeng	2 520	12	0	0	0	0	2 547	12	0	0	0	0	2 564	13	0	0	0	0
462. libquantum	1 110	17	0	0	0	0	1 098	17	0	0	0	0	1 112	17	0	0	0	0
464. h264ref	5 626	20	0	0	0	0	5 636	20	0	0	0	0	5 639	20	0	0	0	0
471. omnetpp	3 491	14	0	0	0	0	4 097	16	0	0	0	0	4 776	18	0	0	0	0
473. astar	1 840	3	0	0	0	0	1 831	3	0	0	0	0	1 807	3	0	0	0	0
483. xalancbmk	13 330	24	0	0	0	0	10 525	22	0	0	0	0	13 567	24	0	0	0	0

SS: single head and single tail; SM: single head and multiple tails; MS: multiple heads and single tail; N: nest; O: overlap; R: reuse

2 标志位的模式化翻译

标志位的模式化翻译基本可以分为 2 个步骤: 1)查找和记录标志位模式;2)翻译标志位模式. 其中第 1 步在指令翻译之前进行,通过扫描基本翻译单元(basic translation unit, BTU)[①] 内的指令来查找标志位模式,并记录查找获得的标志位模式;第 2 步在指令翻译时进行,根据第 1 步记录的标志位模式,在翻译模式头和模式尾时采取相应的翻译策略,从而达到模式化翻译标志位的效果.

2.1 查找和记录标志位模式

标志位模式的查找是通过从后向前扫描 BTU 内的每条指令来完成的. 在扫描过程中遇到对标志位进行引用的指令时,就将其视为潜在的模式尾,然后从该指令的前一条指令开始向前寻找匹配的模式头,即定值该潜在模式尾引用的标志位的指令. 当查找成功时就将模式头和模式尾记录下来,以指导后面标志位模式的翻译. 此外,在查找过程中还需要排除可能的多头单尾结构,以及嵌套、交叠和重用等布局的干扰. 上述标志位模式的查找和记录过程如图 5 所示:

```
/* scan every instruction in current BTU btu from back to front */
FOR (tail=last_instruction(btu);tail≥first_instruction(btu);
tail--)
  /* if current instruction does not use any flag,it is not a pattern
  tail */
  IF (tail uses no flag)
    CONTINUE;
  ENDIF
  /* tail is a potential pattern tail,and find pattern head from
  tail */
  FOR (head=tail-1;head≥first_instruction(btu);head--)
    IF (head defines no flag || head uses any flag)
      CONTINUE;
    ENDIF
    IF (flags defined by head≠flags used by tail)
      /* head and tail cannot construct flag pattern */
      BREAK;
    ELSE
      /* head and tail may construct flag pattern,check pattern
      layout */
       IF (there is no nest,adjacent,reuse in pattern (head,
       tail))
         record pattern (head,tail);
         re-compute the needed definitions of flags at head;
       ELSE
         BREAK;
       ENDIF
    ENDIF
  ENDFOR
ENDFOR
```

Fig. 5 Finding and recording of flag patterns.

图 5 标志位模式的查找和记录

① BTU 是二进制翻译进行指令翻译的基本单元,一般是单个基本块,或多个基本块组成的单入多出的超级块. BTU 之间也可能形成标志位模式,但是由于 BTU 可能存在多个前驱或后继,所以这种模式的查找和翻译比较困难. 此外,这种模式在程序中也比较少,因此本文不对其进行模式化翻译.

当前 BTU 内已查找成功的标志位模式被记录在一个标志位模式队列中. 如第 2 节所述,标志位模式化翻译的目标既包含单头单尾的结构,又包含单头多尾的结构. 因此,需要为每个标志位模式设计一个模式尾链 *tail_list* 来记录多个模式尾. 在记录标志位模式(*head*,*tail*)时,首先在标志位模式队列中查找是否存在以 *head* 为模式头的标志位模式. 如果不存在,说明这是一个新的模式,需要在标志位模式队列中添加新的标志位模式以记录 *head* 和 *tail*;否则,说明这是一个单头多尾的模式,只需将 *tail* 插入到已记录的标志位模式的 *tail_list* 中即可. 在记录完标志位模式后,还需要重新计算在模式头 *head* 处需要翻译的标志位定值,以删除那些仅由模式尾对标志位的引用而产生的标志位定值.

经过上述查找和记录过程后,当前 BTU 内的标志位模式信息均保存在标志位模式队列中. 下面就可以在指令翻译时根据这些保存的信息对标志位模式进行翻译.

2.2 翻译标志位模式

在对 BTU 内的每条指令进行翻译时,首先判断它是否为标志位模式队列中某个标志位模式的模式头或模式尾. 如果是,就按照模式头或模式尾的方式对其进行翻译. 由于指令的翻译是按照指令地址顺序进行的,因此模式头的翻译总是在模式尾之前.

2.2.1 模式头的翻译

在翻译标志位模式的模式头时,对那些仅由模式尾对标志位的引用而产生的标志位定值并不进行翻译,而是在模式尾处根据标志位模式组合的具体语义,选择合适的指令进行翻译. 因此,在翻译模式头时就需要保存用于模式尾进行标志位翻译所需要的信息. 如果使用内存来保存这些信息,那么在存取信息时就会引入额外的访存开销. 因此,本文选择使用目标平台上的寄存器来保存这些信息.

标志位的定值一般只需要指令的运算结果. 但是,对于某些标志位,如 IA-32 平台上的 *CF* 和 *OF* 标志位,当定值这些标志位的是 ADD,ADC 等指令时,就需要通过比较运算结果和源操作数来定值标志位. 由于在模式头处使用寄存器来保存信息,因此,如果既保存源操作数又保存运算结果,那么至少需要 3 个寄存器,这将会给寄存器分配带来压力. 为了减小这种压力,需要根据记录的标志位模式的具体组合来决定需要保存的信息:

1) 如果模式尾翻译标志位时需要运算结果,那么就只保存运算结果;

2) 如果模式尾翻译标志位时需要源操作数,那么就只保存源操作数;

3) 如果模式尾翻译标志位时既需要运算结果,又需要源操作数,那么也只保存源操作数,这是因为运算结果可以重新计算获得.

对于单头多尾结构的标志位模式,不同的模式尾翻译标志位时对源操作数和运算结果的需求可能不同,例如一个模式尾需要运算结果,而另一个模式尾需要源操作数. 在这种情况下,也只保存源操作数,因为需要运算结果的模式尾可以重新计算获得运算结果.

在模式头处,用于保存 2 个源操作数的寄存器记作 R_1 和 R_2;用于保存运算结果的寄存器记作 R_1. 如果需要保存的信息是立即操作数或寄存器,例如某个源操作数是寄存器,并且该寄存器并未被模式头和模式尾之间的指令定值,那么只需记录下立即数或源操作数的寄存器号. 否则,就需要生成指令将要保存的信息存放到 R_1 和 R_2 中.

此外,如果除标志位模式的模式尾之外,该模式头定值的标志位还被其他指令引用,那么在翻译模式头时还需要翻译这些标志位的定值. 模式头的翻译基本流程如图 6 所示. 其中,源操作数的保存是在生成模式头的基本操作之前,而运算结果的保存却是在其之后. 这是因为源操作数也有可能是目标操作数,模式头的基本操作可能会修改这个操作数的内容.

```
/* head is pattern head of some flag pattern in current BTU */
fp=search_head (FLAG_PATTERN_LIST,head);
check which type information needs to be saved according to the type of fp;
IF (source operands need to be saved)
  IF (source operands are not register or immediate || (source operands are register && not defined between pattern head and tail))
    generate instructions to save source operands to R1 and R2;
  ENDIF
  record the source of source operands in fp;
ENDIF
generate instructions to simulate the base function of head;
IF (result needs to be saved)
  IF (result is not register || (result is register && not defined between pattern head and tail))
    generate instructions to save result to R1;
  ENDIF
  record the source of result in fp;
ENDIF
IF (flag definitions used by instructions that are not pattern tails need to be translated)
  generate instructions to define these flags;
ENDIF
```

Fig. 6 The process of translating pattern heads.

图 6 模式头的基本翻译流程

2.2.2 模式尾的翻译

模式尾的翻译主要是根据模式头保存的信息和模式的具体类型,选择合适的指令组合进行标志位的翻译,同时根据计算结果完成模式尾的基本操作,如条件分支、条件置位等.模式尾的基本翻译过程如图 7 所示:

```
/* tail is pattern tail of some flag pattern in current BTU */
fp=search_tail (FLAG_PATTERN_LIST,tail);
IF (source operands in pattern head are recorded in fp)
   extract the information from the source of source operands
   recorded in fp;
ELSE
   extract the information from the source of result recorded in fp;
ENDIF
generate appropriate instruction combination according to the
extracted information and the type of fp.
```

Fig. 7 The process of translating pattern tails.

图 7 模式尾的基本翻译流程

在选择指令组合时,可以根据源平台上标志位模式的语义以及目标平台上的指令特点选择合适的指令组合.以源平台 IA-32 和目标平台 MIPS 为例,表 2 给出了一些常见的标志位模式在翻译模式尾时选择的指令组合,这里假设模式尾翻译标志位时使用的信息均来自于前述 2 个固定的寄存器 R_1 和 R_2.表 2 中 *Rtemp* 是临时寄存器,*Rdest* 和 *Rsrc* 分别为模式尾的源操作数和目标操作数,*target* 为模式尾是条件分支指令时的分支目标地址.从表 2 可以看出,对于很多标志位模式,如 CMP_JZ,TEST_JL 等,在翻译模式尾时仅需要目标平台上的一条指令,这有效降低了因翻译标志位而引入的代码膨胀.

下面可以使用标志位的模式化翻译方法对图 1(a)中的代码片段进行翻译.首先在 804a3cb~804a3d1 的 BTU 内进行标志位模式的查找.通过查找,发现该 BTU 内有 CMP_JG 模式,并将其记录在标志位模式队列中.然后对该 BTU 中的指令进行逐条翻译.在翻译模式头 CMP 时,由于模式尾 JG 翻译标志位时需要 2 个源操作数,另一方面,2 个源操作数均为寄存器,并且在模式头和模式尾之间未被定值,因此在模式头处并不需要将它们保存在固定寄存器 R_1 和 R_2 中,只需要记录下模式尾进行标志位计算时的源操作数来源.在翻译模式尾 JG 时,从记录的源操作数来源处取出保存的信息,并按照 CMP_JG 模式选择合适的指令序列进行翻译.

图 8 给出了经过标志位的模式化翻译后生成的指令序列.可以看出,对标志位进行模式化翻译后,源平台上的 2 条标志位定值引用指令仅生成了目标平台上的 3 条指令,如果不算 70e5f668 处的延迟槽指令,标志位的模式化翻译基本实现了这段代码从源平台到目标平台的一对一翻译.

Table 2 Translation of Pattern Tails in Flag Patterns

表 2 几种常见标志位模式中模式尾的翻译

Flag Patterns	Instructions Generated During Translating Pattern Tails
CMP_JG,SUB_JG	slt *Rtemp*,R_2,R_1 bnez *Rtemp*,*target*
CMP_JL,SUB_JL	slt *Rtemp*,R_1,R_2 bnez *Rtemp*,*target*
CMP_JLE,SUB_JLE	slt *Rtemp*,R_2,R_1 beqz *Rtemp*,*target*
CMP_JGE,SUB_JGE	slt *Rtemp*,R_1,R_2 beqz *Rtemp*,*target*
TEST_JG,OR_JG,AND_JG	bgtz R_1,*target*
TEST_JL,OR_JL,AND_JL	bltz R_1,*target*
TEST_JLE,OR_JLE,AND_JLE	blez R_1,*target*
TEST_JGE,OR_JGE,AND_JGE	bgez R_1,*target*
CMP_JZ,SUB_JZ,TEST_JZ,OR_JZ,AND_JZ	beqz R_1,*target*
CMP_JNZ,SUB_JNZ,TEST_JNZ,OR_JNZ,AND_JNZ	bnez R_1,*target*
CMP_CMOVZ,SUB_CMOVZ,TEST_CMOVZ,OR_CMOVZ,AND_CMOVZ	xor *Rtemp*,R_1,*Rzero* movz *Rdest*,*Rsrc*,*Rtemp*
CMP_CMOVNZ,SUB_CMOVNZ,TEST_CMOVNZ,OR_CMOVNZ,AND_CMOVNZ	sltiu *Rtemp*,R_1,1 movz *Rdest*,*Rsrc*,*Rtemp*

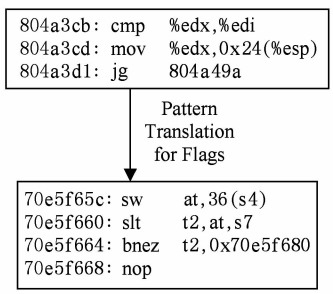


Fig. 8 Pattern translation for flags.

图 8 标志位的模式化翻译

3 实验数据

通过将上述标志位的模式化翻译方法实现在一个已有的二进制翻译系统 DigitalBridge 中,本节将给出对 SPEC CINT2006 进行测试的实验数据.DigitalBridge 是一个动态进程级二进制翻译系统,

它可以在目标平台 MIPS 上实时翻译并执行源平台 IA-32 上的可执行程序.目前 DigitalBridge 支持源平台上各种类型的应用程序,例如科学运算、媒体播放、网络应用等.DigitalBridge 以基本块为 BTU 对程序进行翻译,并在翻译过程中,通过标志位的定值引用分析删除对冗余标志位定值的翻译.

3.1 模式头和模式尾

表 3 列出了 SPEC CINT2006 在输入集 *ref* 下几种常见类型模式头和模式尾的具体数量.从表 3 可以看出,CMP 和 TEST 是模式头中最常见的指令类型.CMP 和 TEST 分别对 2 个源操作数进行算术比较和逻辑比较,然后根据比较结果置各个标志位.它们也是源平台上专门用来定值标志位的指令.除了这 2 类指令外,算术运算指令 SUB,ADD 和 INC 以及逻辑运算指令 OR 和 AND 也是常见类型的模式头.

Table 3 The Number of Common Pattern Heads and Tails

表 3 常见模式头和模式尾的数量

Applications	Pattern Heads								Pattern Tails		
	CMP	SUB	TEST	OR	AND	DEC	ADD	INC	J*cc*	CMOV*cc*	SET*cc*
400.perlbench	7 380	86	6 824	27	318	5	182	25	13 978	440	456
401.bzip2	1 459	27	700	2	24	0	28	16	2 143	89	38
403.gcc	22 341	290	16 189	840	269	2	286	21	37 421	1 026	1 455
429.mcf	704	28	554	1	26	6	41	32	1 340	33	31
445.gobmk	8 569	40	6 012	9	88	6	39	23	13 641	426	1 076
456.hmmer	1 212	52	1 013	9	36	6	67	43	2 412	86	49
458.sjeng	1 532	26	869	10	47	4	37	25	2 388	106	83
462.libquantum	502	20	470	24	26	3	33	18	1 042	30	53
464.h264ref	3 386	58	3 022	47	45	2	134	22	5 659	597	438
471.omnetpp	2 105	38	2 373	3	41	5	67	67	4 457	203	100
473.astar	1 004	43	642	2	30	2	42	18	1 657	81	39
483.xalancbmk	6 588	142	6 600	7	61	3	60	18	12 905	456	208

模式尾中最常见的指令是 J*cc* 类指令,这类指令根据标志位完成条件分支:如果标志位符合条件,就跳转到分支目标;否则不进行任何跳转.除了 J*cc* 类指令外,条件移动类指令 CMOV*cc* 和条件置位类指令 SET*cc* 也是常见类型的模式尾,这些指令也是根据标志位取值的不同进行不同的操作.

图 9 给出了输入集 *ref* 下单头单尾结构的标志位模式中模式头与模式尾相邻和不相邻的相对比例.从图 9 可以看出:一方面,大部分模式头和模式尾均是相邻的;另一方面,模式头和模式尾不相邻的标志位模式还是有一定数量的.

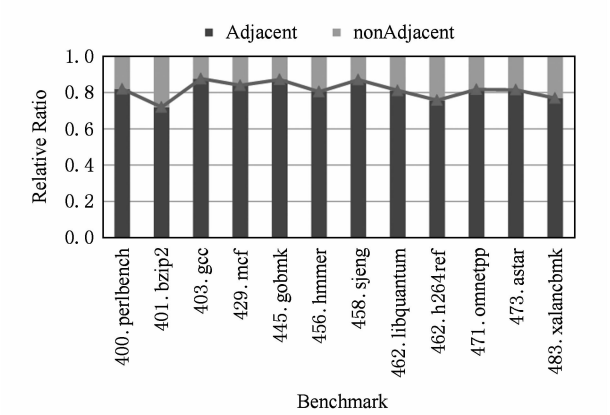


Fig. 9 The relative ratio of the adjacent and nonadjacent flag patterns.

图 9 模式头与模式尾相邻和不相邻的相对比例

3.2 代码量

衡量指令翻译质量的重要指标之一就是翻译生成的代码量.这里代码量是指在翻译执行被翻译程序的过程中,生成的目标平台上的指令总条数.虽然经过标志位分析后,DigitalBridge 在翻译那些会被引用的标志位定值时基本达到了最小代码量,但是标志位的模式化翻译仍然可以进一步减少代码量.图 10 给出了模式化翻译标志位减少的代码量占原始代码量的百分比.可以看出,标志位的模式化翻译在各个输入集下平均减少的代码量达 7.5%.这对于提升二进制翻译的翻译质量、降低翻译生成代码的膨胀有很重要的意义.

模式化翻译标志位不仅可以减少静态的代码量,还可以减少动态执行的指令数.图 11 给出了标志位模式化翻译减少的动态执行指令数.其中,456.hmmer 在输入集 *train* 和 *ref* 下,动态执行的

指令数减少达 30%;429.mcf 在各输入集下动态执行的指令数减少达 20%.平均来看,在各输入集下,标志位的模式化翻译减少的动态执行指令数达 11%.

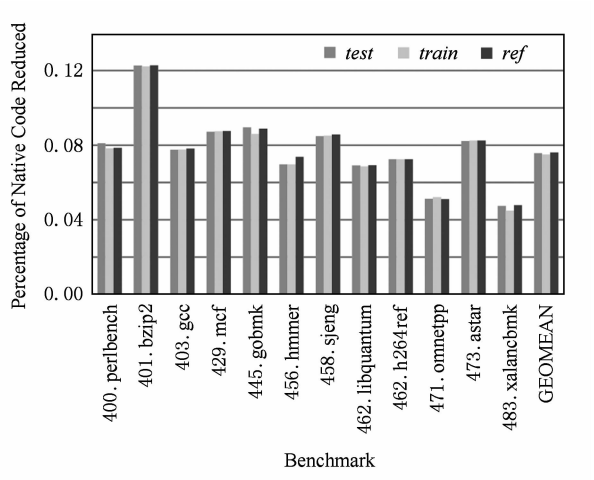


Fig. 10 Percentage of native code reduced.

图 10 标志位的模式化翻译减少生成的代码量

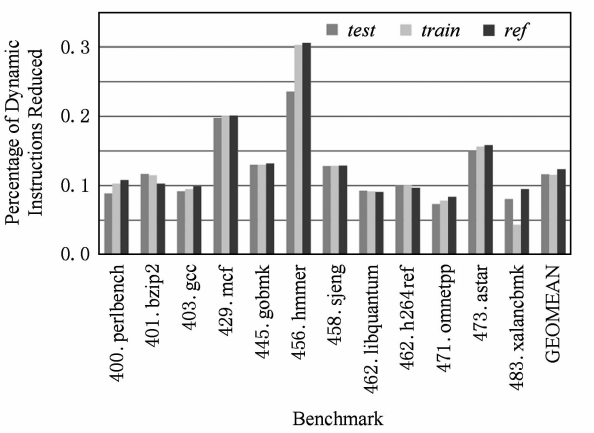


Fig. 11 Percentage of dynamic instructions reduced.

图 11 模式化翻译减少的动态指令数

3.3 性 能

标志位的模式化翻译可以有效降低标志位翻译的开销,提升二进制翻译系统的性能.图 12 给出了标志位模式化翻译带来的性能提升.基本上对每个程序,该方法均带来了性能提升.其中 456.hmmer 在输入集 *train* 和 *ref* 下性能提升达 45%左右,445.gobmk 在各输入集下性能提升达 20%左右.平均来看,标志位的模式化翻译带来的性能提升在各输入集下均达到 10%.

图 13 给出了标志位模式化翻译减少的一级指令 Cache 缺失数.平均来看,在输入集 *test* 和 *train* 下,模式化翻译标志位减少的指令 Cache 缺失数达 10%左右;在输入集 *ref* 下减少达 20%.

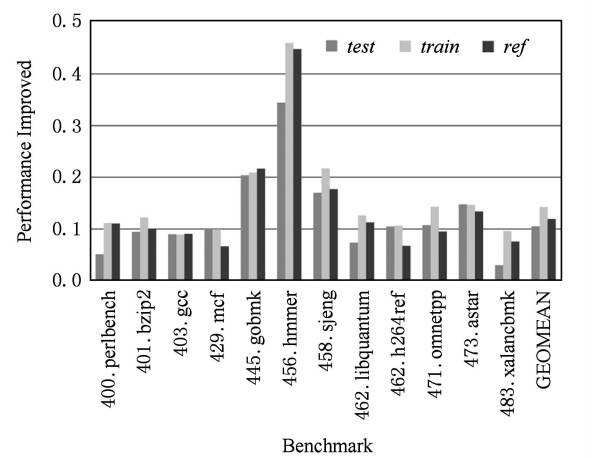


Fig. 12 Performance improved by flag pattern translation.

图 12 标志位模式化翻译带来的性能提升

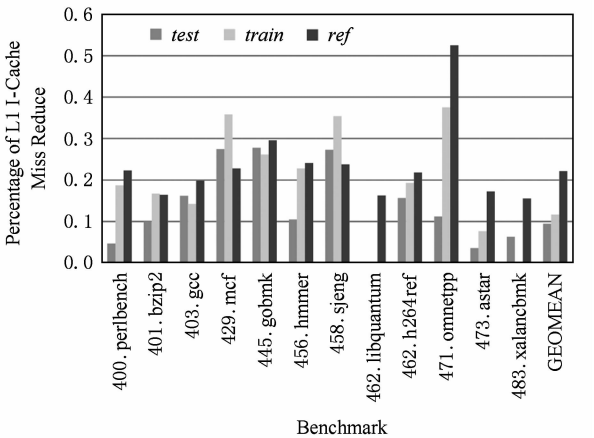
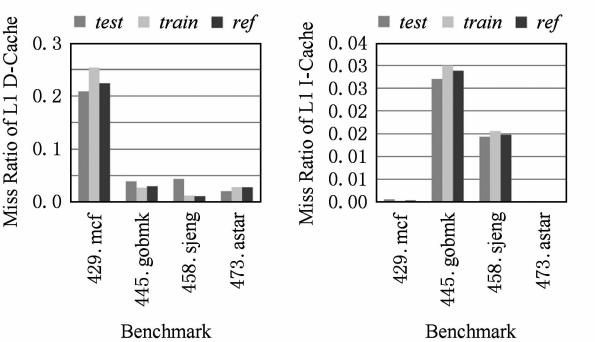


Fig. 13 Percentage of L1 I-Cache miss reduced.

图 13 模式化翻译减少指令 Cache 缺失数

一般来说,动态执行指令数减少较多的程序,其性能提升也应该较高.然而,429.mcf 和 473.astar 却是例外.与 445.gobmk 和 458.sjeng 相比,前两者的动态执行指令数减少比例较后两者大,但是前两者的性能提升却比后两者小.图 14 给出了这 4 个程



(a) Miss ratio of L1 D-Cache (b) Miss ratio of L1 I-Cache

Fig. 14 Miss ratio of L1 D-Cache and I-Cache.

图 14 一级数据和一级指令 Cache 缺失率

序的一级数据 Cache 缺失率和一级指令 Cache 缺失率:

通过对比图 14 中的数据可以看出,429.mcf 的数据 Cache 缺失率较高,由于模式化翻译标志位无法直接减少数据 Cache 的缺失,所以 429.mcf 的性能提升较低.而对于 473.astar,虽然其数据 Cache 缺失率不高,但其指令 Cache 缺失率非常低.因此减少动态执行指令数,对其性能提升的比例没有 445.gobmk 和 458.sjeng 那么高.

标志位的模式化翻译需要对标志位模式进行查找和记录,因此会引入翻译时的性能开销.图 15 给出了具体的性能开销数据.可以看出,在输入集 *test* 下的性能开销要远大于输入集 *train* 和 *ref*,这是因为输入集 *test* 运行时间较短,翻译时间所占比例较大.平均来看,标志位模式化翻译在各输入集下带来的性能开销均在 1%以内.

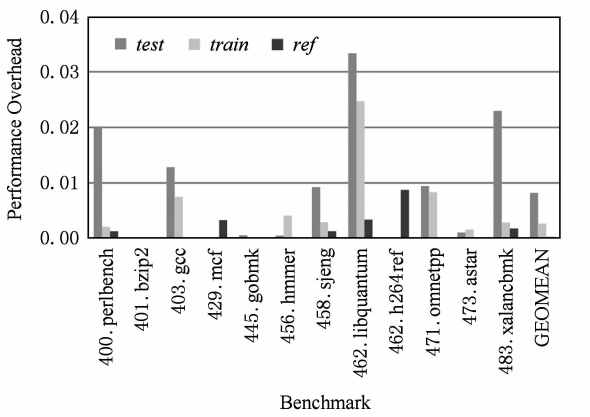


Fig. 15 Performance overhead of flag pattern translation.

图 15 标志位模式化翻译的性能开销

4 相关工作

当前主流的跨平台二进制翻译系统均对标志位的翻译提出了标志位分析技术.这些分析技术的主要目标就是消除二进制翻译中冗余的标志位定值.实际上,如果某条指令定值的某个标志位在后续指令中没有被引用,或者在被引用之前又被重新定值,那么这条指令对该标志位的定值就是冗余的定值.消除这些冗余的标志位定值,对降低二进制翻译系统的开销、提升二进制翻译系统的性能有很大帮助.标志位分析技术通过对标志位进行定值引用分析,建立各个标志位的定值引用关系,从而消除冗余的标志位定值.

FX!32[3]系统使用延迟计算的方法来消除冗余的标志位定值.在执行定值标志位的指令时并不实际计算标志位,而是将计算标志位所需要的信息(包括操作码、2 个源操作数及运算结果)先存储起来,然后当执行到引用标志位的指令时,再根据存储的信息计算标志位.虽然这种延迟计算的方法和本文提到的在模式头保存信息,在模式尾进行标志位翻译有些类似,但是这种方法需要保存的信息较多,并且会带来额外的性能开销.此外,该方法在计算标志位时也没有利用标志位定值引用的具体语义.

QEMU[10]系统使用和 FX!32 类似的延迟计算方式来消除冗余的标志位定值.与 FX!32 不同的是,QEMU 在定值标志位指令处保存的用于计算标志位的信息仅有一个操作数,以及操作码和运算结果.这是因为在计算标志位时,可以根据已保存的信息恢复另外一个操作数.QEMU 通过对当前翻译的基本块进行标志位的定值引用分析,删除那些基本块内的冗余标志位定值,以减少需要保存的信息.

Intel 的 IA32 EL[4]是一个从 IA-32 平台到 Itanium 平台的二进制翻译系统.它首先在热路径(包含多个基本块)上构建不完全数据流图,然后在此数据流图上进行标志位的定值引用分析.这种方法可以有效消除热路径内冗余的标志位定值,但是在热路径的边界上可能会存在无法消除的冗余定值.

文献[5-6]分别提出了 2 类标志位分析算法.其中文献[5]提出的是即时计算与延迟计算(instant computing and delayed computing, ICDC)相结合的算法和数据流分析与延迟计算(data flow analysis and delayed computing, DFADC)相结合的算法,虽然这 2 种算法开销很小,但是仅可以消除基本块内的冗余标志位定值.文献[6]在此基础上作了进一步改进,提出了基于动态反馈的标志位线性分析算法,该算法可以在基本块之间进行标志位的定值引用分析,消除跨基本块的冗余标志位定值.

Harmonia[11]是一个以 IA(intel architecture)为目标平台的二进制翻译器.Harmonia 通过冗余比较删除(redundant-compare elimination)和冗余标志位删除(dead CC-flags elimination)来减少标志位翻译的开销.此外,Harmonia 还通过在目标平台的 IA 指令集中添加新型算术运算指令来进一步降低翻译标志位的开销.同原来的算术运算指令相比,这些新添加的指令不定值 IA 平台上 EFLAGS 寄存器中的标志位.

上述标志位分析方法[4-6,10-11]的共同点就是通

过数据流分析,最大程度地消除冗余的标志位定值.但是,对于那些被引用的标志位定值,这些方法仍然需要对其进行翻译.与上述标志位分析方法不同,本文提出的是一种全新的模式化翻译标志位的方法.该方法可以有效挖掘标志位定值和引用之间的语义关系,并选择目标平台上具有相同语义功能的指令组合对其进行翻译,从而减少翻译会被引用的标志位定值.

文献[12]通过在特定处理器 Loongson 上添加和源平台 IA-32 类似的计算标志位的指令语义,降低二进制翻译系统中翻译标志位的开销.例如,在指令集中添加"X86ADD"指令,该指令可以在进行操作数相加的同时,实现计算标志位的功能.当二进制翻译系统进行标志位翻译时,可以使用这些具有标志位计算功能的指令来简化翻译.这种方法通过在特定处理器上添加额外的硬件支持来提升二进制翻译系统的性能.

文献[13-14]提出了一种基于 edge profile 的热路径选择算法,并在热路径上实现了标志位模式匹配和标志位延迟计算的优化技术,其中标志位模式匹配通过简单的模式匹配将相邻的(CMP|TEST)/Jcc 指令作为指令组合进行整体翻译.与这种简单的模式匹配方法不同,本文提出的模式化翻译方法不仅可以处理相邻的标志位模式,还可以处理不相邻的标志位模式,同时本文提出的方法还可以处理单头多尾结构的标志位模式.此外,本文提出的方法能够处理的模式头和模式尾类型也更加丰富.这些都是文献[13-14]中提到的方法无法做到的.

文献[15]设计了一种将 ARM 平台可执行程序翻译到类 MIPS 平台的静态二进制翻译器.它通过将源平台上的每个标志位映射到目标平台上的单个寄存器来降低翻译标志位的开销.该方法仅适用于源平台标志位较少的情况,对于那些标志位较多的源平台,例如 IA-32 平台,如果目标平台没有足够的寄存器,该方法将无法使用.本文提出的模式化翻译方法仅在翻译模式头时使用 2 个寄存器,在不进行标志位模式的翻译时,这 2 个寄存器还可以参与寄存器分配,因此本文的方法没有上述局限性.

5 结 论

本文提出了一种二进制翻译中标志位的模式化翻译方法,该方法在标志位分析技术的基础之上,将源平台上标志位的定值指令和引用指令组合成固定的标志位模式,并根据标志位模式的具体语义选择目标平台上合适的指令组合进行翻译.这样就可以进一步减少对那些被引用的标志位定值的翻译.实验结果表明,这种模式化翻译标志位的方法不仅可以降低翻译标志位引入的代码膨胀,还可以有效提升二进制翻译系统的性能.

参 考 文 献

[1] Luk C, Cohn R, Muth R, et al. Pin: Building customized program analysis tools with dynamic instrumentation [C] // Proc of PLDI'05. New York: ACM, 2005: 190-200

[2] Nethercote N, Seward J. Valgrind: A framework for heavy weight dynamic binary instrumentation [C] //Proc of PLDI'07. New York: ACM, 2007: 89-100

[3] Chernoff A, Herdeg M, Hookway R, et al. FX!32—A profile-directed binary translator [J]. IEEE Micro, 1998, 18(2): 56-64

[4] Baraz L, Devor T, Etzion O, et al. IA-32 execution layer: A two-phase dynamic translator designed to support IA-32 applications on Itanium®-based systems [C] //Proc of MICRO'03. Piscataway, NJ: IEEE, 2003: 191-201

[5] Ma Xiangning, Wu Chenggang, Tang Feng, et al. Two condition code optimization approaches in binary translation [J]. Journal of Computer Research and Development, 2005, 42(2): 329-337 (in Chinese)
(马湘宁, 武成岗, 唐峰, 等. 二进制翻译中的标志位优化技术[J]. 计算机研究与发展, 2005, 42(2): 329-337)

[6] Tang Feng, Wu Chenggang, Feng Xiaobing, et al. EfLAalgorithm based on dynamic feedback [J]. Journal of Software, 2007, 18(7): 1603-1611 (in Chinese)
(唐锋, 武成岗, 冯晓兵, 等. 基于动态反馈的标志位线性分析算法[J]. 软件学报, 2007, 18(7): 1603-1611)

[7] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture [EB/OL]. [2013-01-02]. http://download.intel.com/products/processor/manual/253665.pdf

[8] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined, Volumes 2A, 2B, and 2C: Instruction Set Reference, A-Z [EB/OL]. [2013-01-02]. http://download.intel.com/products/processor/manual/325383.pdf

[9] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined, Volumes 3A, 3B, and 3C System Programming Guide, Parts 1 and 2 [EB/OL]. [2013-01-02]. http://download.intel.com/products/processor/manual/325384.pdf

[10] Bellard F. QEMU, a fast and portable dynamic translator [C] //Proc of USENIX ATC'05. Berkeley: USENIX Association, 2005: 41-46

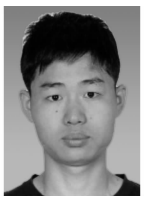
[11] Ottoni G, Hartin T, Weaver C, et al. Harmonia: A transparent, efficient, and harmonious dynamic binary translator targeting the Intel® architecture [C] //Proc of ACM CF'11. New York: ACM, 2011: 1-10

[12] Hu Weiwu, Liu Qi, Wang Jian, et al. Efficient binary translation system with low hardware cost [C] //Proc of IEEE ICCD'09. Piscataway, NJ: IEEE, 2009: 305-312

[13] Bai Tongxin, Feng Xiaobing, Wu Chenggang, et al. Optimizing dynamic binary translator in DigitalBridge [J]. Computer Engineering, 2005, 31(10): 103-105 (in Chinese)
(白童心,冯晓兵,武成岗,等. 优化动态二级制翻译器DigitalBridge[J]. 计算机工程, 2005, 31(10): 103-105)

[14] Bai Tongxin. Two topics on dynamic binary translation and optimization [D]. Beijing: Institute of Computing Technology, Chinese Academy of Sciences, 2004 (in Chinese)
(白童心. 动态二进制翻译与动态优化相关问题研究[D]. 北京:中国科学院计算技术研究所, 2004)

[15] Chen J Y, Wang W, Huang T H, et al. On static binary translation and optimization for ARM based applications [C] //Proc of the 6th Workshop on Optimizations for DSP and Embedded Systems. New York: ACM, 2008: 1-10



Wang Wenwen, born in 1986. PhD candidate. His main research interests include dynamic optimization and binary translation.

Wu Chenggang, born in 1969. PhD and associate professor. Senior member of China Computer Federation. His main research interests include dynamic optimization and binary translation (wucg@ict.ac.cn).

Bai Tongxin, born in 1977. PhD. His main research interests include compiler, parallel programming, and big data analytics (waffle.bai@gmail.com).

Wang Zhenjiang, born in 1983. PhD and assistant professor. His main research interests include dynamic optimization and binary translation (wangzhenjiang@ict.ac.cn).

Yuan Xiang, born in 1984. PhD candidate. His main research interests include dynamic optimization and binary translation (yuanxiang@ict.ac.cn).

Cui Huimin, born in 1979. PhD and associate professor. Her main research interests include programming model and compilers (cuihm@ict.ac.cn).