计算机研究与发展
Journal of Computer Research and Development

ISSN 1000-1239/CN 11-1777/TP
43(12): 2174~2179, 2006

二进制翻译中的库函数处理

杨 浩 唐 锋 谢海斌 武成岗 冯晓兵
(中国科学院计算技术研究所 北京 100080)
(yanghao1102@ict.ac.cn)

Library Function Disposing Approach in Binary Translation

Yang Hao, Tang Feng, Xie Haibin, Wu Chenggang, and Feng Xiaobing
(*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100080)

Abstract Disposing the library functions call fast and efficiently is an important performance issue in the binary translation technology. One algorithm named JLSCL (jacket library and shortcut library) is presented to dispose the library functions classified, which is based on the dynamic binary translator integrated with static pre-translator. JLSCL can make use of the merit of dynamic translation and static translation, and make use of the convention of the function call on the target processor to reduce the redundant memory access. These make it more efficient that the original binary code runs on the target processor. The algorithm can flexibly switch between the library function and system call, and have applicability for the library function. The algorithm is verified in the binary translator system—digital bridge version 2. It can dispose the library function call successfully and efficiently, and its performance has been improved greatly.

Key words binary translation; library function dispose; jacket library function; PLT-shortcut library function

摘 要 在二进制翻译技术中,快速有效地处理系统库函数调用是一个值得研究的课题.基于动静结合二进制翻译技术,提出一种对系统库函数进行分类处理的算法,能够很好地利用动静结合二进制翻译的技术特点和目标机本地的函数调用约定,减少不必要的内存操作,提高源二进制代码在目标机上的执行效率.该算法能够在系统库函数和系统调用之间灵活地切换,并且对于系统库函数有较强的适用性.经过大量的测试验证,在应用该算法后,Digital Bridge Version 2 系统不仅能够正确有效地处理系统库函数调用,并且性能有了很大的改善.

关键词 二进制翻译;库函数处理;可包装库函数;PLT 短路库函数

中图法分类号 TP314

二进制翻译技术是目前解决软件移植问题的一个研究热点,能将现有的软件移植到新开发的处理器上执行,其目的不仅仅是仿真源指令集体系结构(instruction set architecture, ISA),更高的目标是翻译产生的代码在目标 ISA 上运行的速度等同甚至超过源 ISA 上的原有代码.

基于软件的二进制翻译分为动态和静态两种翻译形式[1],动态翻译对于间接指令可以知道实际的跳转目标,而静态翻译可以利用传统静态编译优化方法.动静结合二进制翻译技术综合了两者的优点,优化以静态为主,避免动态优化本身的时间消耗.动态通过翻译执行本地码,把静态不能识别的间接

收稿日期:2005-12-02;**修回日期**:2006-03-29
基金项目:国家自然科学基金项目(60403017)

入口点传给静态帮助其扩大翻译优化的范围.经过动静多遍迭代,最后几乎所有的本地代码都由静态翻译产生.

二进制翻译中一类较重要的问题就是处理源二进制代码中出现的库函数调用.不同ISA系统平台上传递参数和返回值的方式有较大的差异,为了消除这种差异必然会带来执行效率的下降.由于编程上的模块化和代码重用思想,二进制代码中会较为频繁地调用系统库函数,尤其是具有实际应用价值的大程序.因此,对库函数调用的处理既关系到程序执行的正确性,又涉及到翻译后代码的执行效率.

本文提出了一种基于动静结合二进制翻译技术的库函数处理算法(jacket library and shortcut library, JLSCL),将库函数分为可包装库函数和PLT短路库函数两类,对分类后的库函数采用两种有效的算法JLD(jacket library disposing)和SCLD(shortcut library disposing)分别处理,消除不同ISA之间库函数调用的差异性,同时兼顾执行速度和管理维护的便捷性.

1 相关工作

二进制翻译系统处理库函数调用所采用的方法与各自系统的特点有关.

FX!32[2]是DEC研发的轮廓制导的二进制翻译器,结合了解释执行和静态翻译两种翻译模式,能将运行在(X86,WinNT)系统下的应用程序运行在(Alpha,WinNT)下.它主要是用内存模拟X86上的堆栈及其参数压栈和取返回值的行为,对大约12000个Win32 API进行包装,一些包装还要对被调用程序语义进行修改,同时也对COM组件进行了包装.

Bintrans[3]是应用级动态二进制翻译器.主要是对系统调用进行包装.用内存模拟源机器的参数传递行为,并且还要消除源与目标机器在系统调用上的语义差别.因为处理系统调用,因此相比处理库函数减少了很多工作量,但是也因此降低了性能.

VEST[4]和mx[4]是DEC开发的两个二进制翻译器,目的是为了使已有的VAX和MIPS代码能够在新的Alpha AXP机器上运行.翻译界面定在需要翻译的主文件和系统库之间,模拟源二进制程序所在机器的函数调用约定,对本地库函数进行自动包装处理,但一些特殊的库函数包装要手写,修改执行语义来消除源与目标机器平台的差异.

IA-32 Execution Layer[5]是Intel公司开发的动态二进制翻译器,能够把应用级的ia32程序翻译到ia64体系结构上,没有跨操作系统,能在Windows和Linux上运行.为了能够在多平台上运行,IA-32 EL主要由两个模块搭建而成,操作系统无关的模块BTGeneric,存放系统调用包装的模块BTLib,这两个模块之间通过约定好的API进行通信.

UQDBT[6]和Walkabout[7]分别是Queensland大学和Sun公司开发的可变源和可变目标的动态二进制翻译器.UQDBT是对库函数进行包装来处理系统库函数调用,对一些特殊的库函数要手写修改包装.Walkabout是对系统调用进行包装处理,这样避免很多特殊库函数的手写工作.

Transitive公司推出的商用化产品QuickTransit,也是采用可变源和可变目标的二进制翻译技术,这个公司直到今天仍然致力于二进制翻译技术的研究,具体的相关信息可以参看公司的主页http://www.transitives.com/.

2 背景知识介绍

由于不同ISA系统平台的对齐方式存在差异以及翻译界面的定制问题,将引起很多难以处理的问题,在论述算法之前将介绍处理过程中遇到的问题和翻译界面的定制.

2.1 处理库函数调用过程中遇到的问题

二进制翻译在处理库函数调用过程中,主要有两个问题比较难以处理:

1)结构变量对齐问题.由于不同ISA系统平台对于某些数据类型的对齐方式可能不同,因此一个结构变量的某些域变量相对于结构变量起始存储位置有不同的偏移,当库函数要访问这个结构变量时,就可能发生域变量的错误引用;或者由于不同ISA的同名结构变量中域变量的个数不同,因此源ISA结构变量中的某个域可能在目标ISA上根本没有,反之亦然;或者由于库函数A调用库函数B,A中没有前面这两种对齐问题,而B中存在,因此也会出现域变量错误引用的问题,这时我们也认为A存在对齐问题.当然,这些还只是在源与目标ISA大小尾相同情况下的对齐问题,若不同,还要进行一定的转换来消除大小尾的差异.

2)同名全局变量在不同文件有多个副本.一个可执行程序在执行时,重定位技术会保证同名全局变量在整个程序执行过程中只有一个副本,其他文

件中若出现这个变量,会在全局偏移表(global offset table,GOT)中指向变量副本的实际位置.但当源ISA上的可执行程序通过二进制翻译技术在目标ISA上执行时,这个可执行程序与它所涉及的系统库可能分属于不同的ISA,因此无法保证全局变量副本的统一,这样就可能引用错误的全局变量值.这两个问题都涉及到二进制翻译界面的定制问题,是二进制翻译技术在处理库函数调用时不可回避的问题.

2.2 翻译界面的定制

这里所说的翻译界面是指归属于源ISA的文件和归属于目标ISA的文件之间的分界线.由上面对库函数处理中遇到问题的分析可知,翻译界面若定在需要翻译的主文件和共享系统库之间,就会引入大量的对齐问题和全局变量多副本问题,全局变量多副本在这种情况下可能是无法消除的.而定在系统调用一级,源机器的库函数代码也是被翻译的对象,同名全局变量多副本问题将不再出现,虽然在系统调用中也可能存在结构变量对齐问题,但是Linux操作系统中,系统调用总共只有200多个,这在实现和管理上会带来很大的便捷.

3 库函数处理算法设计

首先介绍两个在算法中使用的定义.

定义1. 可包装库函数.如果某个库函数没有同名全局变量多副本和结构变量对齐差异这两类问题,那么可以调用目标机本地的库函数,不会因为翻译过程链接表(procedure linkage table,PLT)、动态链接器、库函数的源二进制代码造成代码膨胀导致的性能损失.

定义2. PLT短路库函数.有些库函数在静态阶段已经翻译过,动态阶段可以直接跳入静态本地码中,不需要翻译执行PLT和动态链接器,相当于把动态链接在某种程度上还原回静态链接,也避免了因为翻译PLT、动态链接器而造成的性能损失.有些库函数虽然在初始的静态阶段没有被翻译,但可以采用回填的方法在库函数代码被翻译之后填写目标地址,这种情况也可以认为是PLT短路库函数,在后面的算法设计中有讲解.

3.1 JLSCL算法概述

目前库函数处理的方法主要有两种:一种是对本地库函数的包装[2,4,6],在目标机上模拟源机器的传递参数和返回值的行为,通过包装与本地库函数之间传递参数和返回值,但存在前面所述的多个副本以及对齐差异的问题;另一种是包装目标机器的系统调用[3,5,7],按照源ISA处理动态链接的执行流程,把PLT、动态链接器、库函数的源二进制代码都进行翻译执行,会造成较大的代码膨胀,导致执行效率的下降.

为了克服这两种方法的缺点,JLSCL算法采用对库函数分类的思想,分为可包装库函数和PLT短路库函数.区分的依据主要是看是否有全局变量多副本和结构变量对齐方式差异这两类问题,优先考虑一个函数是否是可包装库函数,若不是再考虑是否是PLT短路库函数,若两种都不是,则采用上面的第2种方法保守处理.

本文算法建立在如下3个基本的原则上.①尽可能地识别和提升源ISA二进制程序的库函数调用;②对可包装库函数尽可能地避免模拟源ISA的函数调用约定[8,9],而是用目标机本地函数调用约定[10]来处理库函数调用,从而提高翻译出来的本地码的执行效率;③对于PLT短路库函数尽可能地利用静态模块翻译的质量较高的本地码来避免翻译PLT和动态链接器的源ISA代码,从而提高翻译出来的本地码的执行效率.

3.2 JLSCL算法设计

3.2.1 JLSCL算法

算法初始化时,对涉及到的系统库函数建立分类查询表;在静态阶段对主文件及其依赖的共享库中的系统库函数进行翻译;在控制转移到动态翻译执行之前,完成动态阶段的装入工作,需要装入内存的内容包括源ISA的主文件和系统库,静态阶段翻译出来的本地码,包括主文件本地码和系统库本地码;在目标机上用内存模拟源ISA的寄存器和堆栈.

动态翻译执行过程中以源ISA二进制代码中的每个基本块为单位,按照程序的逻辑执行顺序依次进行处理,当处理某个源ISA的基本块时,判断基本块的结束指令的指令类型是否为CALL指令.

```
若结束指令为CALL指令
{
  若CALL的目标是库函数
  {
    遍历库函数分类查询表,判断库函数的类型;
    若是可包装库函数,则按照JLD算法处理;
    若是PLT短路库函数,则按照SCLD算法处理;
```

}

否则模拟源 ISA 传递参数和返回值的约定,对这种主文件内部的普通函数调用进行处理.

}

当按照逻辑流程处理完所有的源 ISA 的基本块后,结束源二进制代码的翻译执行.

3.2.2 JLD 算法

JLD 算法对库函数采用包装的方法处理,可以采用内存来模拟源 ISA 的寄存器和堆栈空间,但这样就会有更多的内存操作,因此应该尽可能利用目标机本地函数调用约定来处理库函数. 首先根据库函数的形参说明,在当前基本块内向前寻找参数压栈指令.

若查找成功

{

把基本块内为了翻译参数压栈指令而生成的本地码,修改成把参数按照目标机 ABI 的传参约定放入到相应的寄存器和堆栈的本地代码;

按照源机器 ABI 的约定,将堆栈进行还原,以 X86 为例,进行如下变化:

X86 模拟寄存器 ESP = X86 模拟寄存器 ESP - 所有参数总大小;

调用本地库函数;

按照目标机传返回值约定取出返回值,然后按照 X86 约定放入到 X86 模拟寄存器或内存中,并结束对当前基本块的处理;

}

否则

在目标机上模拟源机器参数和返回值传递约定,调用包装的本地库函数完成库函数调用的处理,并结束对当前基本块的处理.

3.2.3 SCLD 算法

SCLD 算法是希望尽快利用静态翻译出来的高质量本地码,首先要在已经翻译的本地码中查找当前要处理的库函数的本地码,对于查找不成功的情况采用回填技术处理.

若查找成功

{

把调用库函数的 CALL 指令翻译成如下的本地代码:跳转到库函数本地码在内存中存放的地址;

把返回地址放入约定好的寄存器中;

}

否则

{

生成跳转目标是空的转移指令,并做好标记;

把返回地址放入约定好的寄存器中;

当内存中有库函数的本地码之后,再把本地码的存放地址填入跳转指令中.

}

3.3 JLSCL 算法分析

JLSCL 算法由 JLD 算法和 SCLD 算法组成. JLD 算法调用本地机器优化过的系统库函数,并且尽可能利用目标机本地的函数调用约定,来减少因为模拟源机器传递参数和返回值造成的不必要的内存操作;而 SCLD 算法主要利用动静结合的优势,在动态执行时调用静态阶段翻译优化过的源机器系统库函数,虽然静态模块可以对源 ISA 系统库函数的翻译代码进行深度优化,但静态优化只能使翻译所造成的代码膨胀得到一定程度的减弱,不能真正消除. 因此 JLSCL 算法中的 JLD 算法带来的性能提高强于 SCLD 算法.

由于库函数处理过程中存在第 2.1 节中论述的两类问题,JLD 算法不具有普适性,对于 Linux 系统中近 10 万个系统库函数,不是所有的库函数都是可包装库函数,而且如果把所有的库函数都包装,工作量也是非常巨大的;SCLD 算法虽然不如 JLD 算法对于性能提高的贡献大,但具有较强的普适性. 因为只要某个库函数在静态阶段被翻译优化过或动态阶段通过回填的方式翻译过,那么它就是 PLT 短路库函数,而且 SCLD 算法只需要对操作系统内核提供的 200 多个系统调用进行包装,在编程实现和管理维护上很方便. 因此,JLD 算法和 SCLD 算法的结合既能带来很好的性能,又能具有较强的普适性并且管理维护便捷.

JLSCL 算法普遍适用于采用动静结合方式的二进制翻译,很好地利用了动静结合的技术特点和目标机本地的函数调用约定,减少了内存的操作次数,有利于提高二进制翻译的性能.

4 算法实现和验证

JLSCL 算法已经在我们开发的 Digital Bridge Version 2 系统中得到实现,并且通过了大量测试用例的验证,说明该算法不仅是正确的,而且能给系统带来很好的性能改善.

4.1 Digital Bridge Version 2 系统简介

Digital Bridge Version 2 是动静结合的应用级

二进制翻译器,用于将 X86 处理器上的二进制代码翻译成 MIPS 处理器的二进制代码,并且在 MIPS 处理器上即时执行翻译后的代码,翻译界面定在系统调用一级.

本系统由静态翻译器和动态翻译执行器两部分组成. 静态翻译器在程序执行之前,根据 profiling 结果对 X86 代码进行翻译优化,得到静态翻译结果,包括翻译后的 MIPS 本地代码以及一些辅助信息. 动态翻译执行器获取这些静态翻译结果后,将 X86 源二进制代码和静态翻译结果装入内存,并翻译执行源二进制代码,在执行过程中记录 profiling 结果,供静态翻译器下一次翻译时使用. 该系统中动静态翻译交替执行,经过若干次迭代后,被翻译程序趋于稳定,得到质量较高的目标机本地码. 动静结合的流程如图 1 所示:

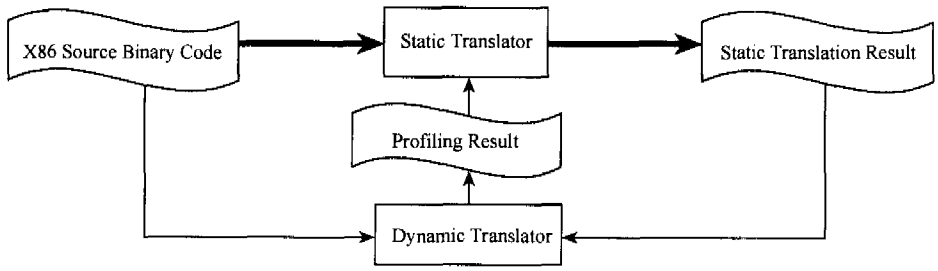

Fig.1 Digital bridge Version 2 process flowgraph.

图 1 Digital bridge Version 2 执行流程图

4.2 测试数据的选择

Digital Bridge Version 2 系统已经通过了 2000 多个 C 语言程序的宽度测试和深度测试(包括约 400 多万行源代码编译生成的 1000 万条以上的 X86 指令),包括若干超过万行 C 语言源码的大型程序如 crafty,这证明了 JLSCL 算法是正确可行的.

为了说明 JLSCL 算法对于性能提高的作用,对 SPEC CPU2000 INT 中部分 benchmark 的 test 测试集进行了测试. 表 1 是对本文所采用的测试程序的特征描述:

Table 1 Benchmark Descriptions

表 1 Benchmark 特征描述

Benchmark	Description
bzip2	Compression
gzip	Compression
crafty	Game playing: Chess
mcf	Combinatorial optimization
twolf	Place and route simulator
vortex	Object-oriented database
vpr	FPGA circuit placement and routing
perlbmk	PERL programming language
gcc	C progaramming language compiler

4.3 实验结果

为了体现 JLSCL 算法的优势,主要测试数据包括未采用 JLSCL 算法时对系统调用进行包装的系统执行时间,以及采用 JLSCL 算法之后的系统执行时间. 只是比较这两者而不考虑和包装库函数的方法相比,是因为包装库函数的方法普适性较差,对比的实际意义不大.

假设变量 t_1, t_2 分别表示包装系统调用和采用 JLSCL 算法的执行时间,实验结果如图 2 所示:

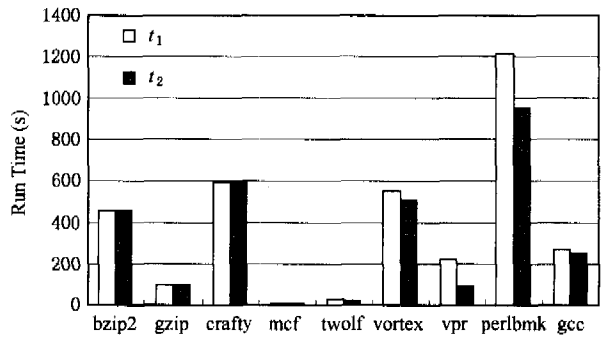

Fig.2 Jacket system call vs. JLSCL.

图 2 两种方法的测试时间对比

从图 2 中可以看出,在应用了 JLSCL 算法之后,系统的执行速度有了一定程度的提高,尤其是 vpr 和 perlbmk. 主要是因为 vpr 和 perlbmk 中对系统库函数的调用多发生在执行频率较高的基本块中,而那些系统库函数调用发生在执行频率不高的基本块的 benchmark 则加速比相对较小,例如 bzip2,gzip 和 crafty,其中 bzip2,gzip 速度几乎没有提高,crafty 甚至出现了速度的负增长,而其余 4 个 benchmark 大约有 10% 的加速比. 这些都说明了 JLSCL 算法具有很好的实效性,能够给 Digital Bridge Version 2 系统带来很好的性能改善.

5 结论和今后的工作

本文提出的 JLSCL 算法利用了动静结合二进制翻译的技术特点,并且利用了目标机本地的函数调用约定,减少了不必要的内存操作,体现了动态与静态之间的互补作用,同时也很好地兼顾了执行速度和管理维护上的便捷性. 该算法经过 Digital Bridge Version 2 系统的正确性和有效性验证,证明可以有效地解决二进制翻译中的库函数处理这类问题.

在处理可包装库函数时,对库函数的包装处理还带有很多的手工痕迹,能否有效地实现自动包装库函数对于二进制翻译系统的编程实现和管理维护很有意义. Transitive 公司一直致力于二进制翻译相关工作的商用化推广,也必将使二进制翻译研究在未来有更广阔的空间.

参 考 文 献

[1] E R Altman, D Kaeli, Y Sheffer. Welcome to the opportunities of binary translation [J]. IEEE Computer, 2000, 33(3): 40-45

[2] R J Hookway, M A Herdeg. Digital FX!32: Combining emulation and binary translation [J]. Digital Technical Journal, 1977, 9(1): 3-12

[3] Mark Probst. Dynamic binary translation [C]. UKUUG Linux Developer's Conf, Bristol, 2002

[4] Richard L Sites, Anton Chernoff, Matthew B Kirk, *et al*. Binary translation [J]. Communications of the ACM, 1993, 36 (2): 197-218

[5] Leonid Baraz, Tevi Devor, Orna Etzion. IA-32 execution layer: A two-phase dynamic translator designed to support IA-32 applications on Itanium(r)-based systems [C]. In: Proc of the 36th Int'l Symp on Microarchitecture (MICRO-36 2003). Los Alamitos: IEEE Computer Society Press, 2003

[6] D Ung, C Cifuentes. Machine-adaptable dynamic binary translation [C]. In: Proc of the ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization. New Yrok: ACM Press, 2000. 30-40

[7] C Cifuentes, B Lewis, D Ung. Walkabout—A retargetable dynamic binary translation framework [C]. The 4th Workshop on Binary Translation, Charlottesville, Virginia, 2002

[8] 243190 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture. Santa Clara, CA: Intel, 1999

[9] 243191 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Referenc. Santa Clara, CA: Intel, 1999

[10] Joe Heinvich. 94039-7311 MIPS R4000 Microprocessor User's Manual, Second edition. Mountain View: MIPS Technologies, Inc, 1994

Yang Hao, born in 1979. Master. His main research interests include binary translation.

杨浩,1979 年生,硕士,主要研究方向为二进制翻译.

Tang Feng, born in 1979. Ph. D. His main research interests include binary translation and compiler optimization.

唐锋,1979 年生,博士,主要研究方向为二进制翻译、编译优化(tf@ict.ac.cn).

Xie Haibin, born in 1980. Ph. D. candidate. His main research interests include binary translation and compiler optimization.

谢海斌,1980 年生,博士研究生,主要研究方向为二进制翻译、编译优化(xhb@ict.ac.cn).

Wu Chenggang, born in 1969. Ph. D. and associate professor, senior member of China Computer Federation. His main research interests include binary translation and compiler optimization.

武成岗,1969 年生,博士,副研究员,中国计算机学会高级会员,主要研究方向为二进制翻译、编译优化(wucg@ict.ac.cn).

Feng Xiaobing, born in 1969. Ph. D., professor, senior member of China Computer Federation. His main research interests include parallel compiling, binary translation and tools.

冯晓兵,1969 年生,博士,研究员,中国计算机学会高级会员,主要研究方向为并行编译技术、二进制翻译、相关工具环境(fxb@ict.ac.cn).

Research Background

Binary translation is a technique used to translate the executable binary code of a computer architecture and operating system into the executable binary code of another computer architecture and operating system.

One of the challenges in binary translation is how to achieve run time performance on target architecture, which is as good as or even better than native code. Disposing the library functions call fast and efficiently in the binary translation technology is an essential method to reduce the redundant memory access and improve the system performance. Working with other optimizations, binary translation can be used as commercial product and save the effort of software development and migration.

This research is sponsored by the Chinese National Science Fund (60403017).