ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, Vol.20, Supplement, December 2009, pp.1−7

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# 二进制翻译中的寄存器映射与剪裁的实现*

文延华+, 唐大国, 漆锋滨

(江南计算技术研究所,江苏 无锡 214083)

# Register Mapping and Register Function Cutting out Implementation in Binary Translation

WEN Yan-Hua+, TANG Da-Guo, QI Feng-Bin

(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

+ Corresponding author: wenyanhua.ok@163.com

**Wen YH, Tang DG, Qi FB. Register mapping and register function cutting out implementation in binary translation**. ***Journal of Software*, 2009,20(Suppl.):1−7.** http://www.jos.org.cn/1000-9825/09001.htm

**Abstract**: How to migrate binary code between different ISA efficiently is a difficult problem in binary translation. This paper analyzes this problem at the aspect of register mapping and an innovative register mapping method combining segment mapping and function cutting out of special-purpose register has been presented in this paper. An implementation of the method has been done on trend, which is a dynamic binary translation system translating and executing binary code of PowerPC on Alpha. The testing results of NPB-serial and SPEC2000 show that the method can simplify instruction translation, decrease code expanding and improve the execution efficiency of binary code obtained after binary translation evidently.

**Key words**: binary translation; register mapping; special-purpose register; segment mapping; function cutting out of special-purpose register

**摘 要**: 如何进行异构机之间可执行程序的高效移植是二进制翻译面对的难点问题.从寄存器映射的角度分析了这一问题,提出了分段映射和特殊寄存器功能剪裁相结合的方法,以 trend 系统为平台进行了实验和测试.NPB-serail 测试包和 SPEC2000 测试包的测试结果显示:使用该方法,可以简化指令翻译,减少代码膨胀,有效提高翻译后代码的执行效率.

**关键词**: 二进制翻译;寄存器映射;特殊寄存器;分段映射;特殊寄存器功能剪裁

二进制翻译是一种在目标平台上运行为源平台编译生成的可执行代码的技术.它提供了无须重新编译源程序就能将可执行代码自动转换到新的体系结构的支持,成为解决软件移植问题的研究热点,得到了广泛重视.HP[1],Transmeta[2],IBM[3],Digital[4],SUN[5],Transitive[6]等知名公司和Queensland大学[7]、Vienna大学等科研机构都在二进制翻译领域进行了深入研究,取得了一系列成果,构建了一批商用二进制翻译系统和几个公开的二进制翻译架构.

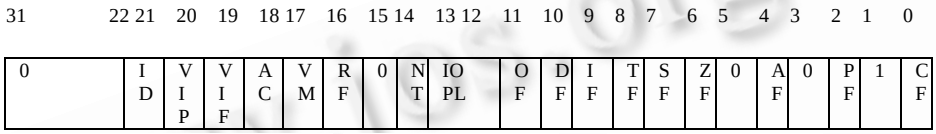
二进制翻译要解决异构机之间的代码移植问题,必须要分析不同体系结构之间的异、同之处,确定不同的数据存储方式、不同的寄存器文件、不同的页面大小之间的映射关系.本文主要分析了异构机之间的寄存器映射问题.

* Supported by the National Basic Research Program of China under Grant No.2007CB310900 (国家重点基础研究发展计划(973))
Received 2008-07-01; Accepted 2009-07-17

## 1 寄存器映射

二进制翻译中的寄存器映射指的是将源结构中的寄存器映射到目的结构的过程.对于一般的通用寄存器和浮点寄存器,采用的映射策略是一一映射,即源结构的可执行程序中使用的每一个通用寄存器被分别映射到目标结构不同的虚通用寄存器,源结构的可执行程序中使用的每一个浮点寄存器被分别映射到目标结构不同的虚浮点寄存器,虚寄存器与实际寄存器的对应问题由二进制翻译系统的寄存器分配模块解决.但是对于有特殊功能的专用寄存器,一一映射的策略有时候会影响翻译后程序的运行效率.下面进行详细的分析说明.

某些架构的机器存在一些特殊寄存器,用于反映运算的结果,并且指导分支的走向.例如X86 架构中的EFLAGS寄存器[8],该寄存器包括一组状态标志、一组控制标志和一组系统标志.图 1 是EFLAGS寄存器的标志位示意图,其中,CF,PF,AF,ZF,SF,OF位属于状态标志位,用来反映算术指令运行结果的状态,加、减、乘、除等算术运算,与、或、异或等逻辑运算,以及比较、移位、旋转等操作都会设置这些状态标志位.控制转换指令会根据这些状态标志位来确定程序的走向.




System flag
ID: ID flag
VIP: Virtual interrupt pending
VIF: Virtual interrupt flag
AC: Alignment check
VM: Virtual-8086 mode
RF: Resume flag
NT: Nested Task
IOPL: I/O privilege level
IF: Interrupt enable flag
TF: Trap flag

Status flag
OF: Overflow flag
SF: Sign flag
ZF: Zero flag
PF: Parity flag
CF: Carry flag
AF: Auxiliary carry flag

Control flag
DF: Direction flag

Fig.1 EFLAGS Register in X86

图 1 X86 的 EFLAGS 寄存器

PowerPC架构中也有类似的特殊寄存器[9],32 位的条件寄存器CR有 8 个域,每一个域 4 位,从低位开始依次为CR0,CR1,…,CR7,如图 2 所示.PowerPC中整型运算指令后带‘.’,例如“add.”,说明除了进行普通运算外,还需要根据目标寄存器的值对CR寄存器的CR0 域赋值,CR0 域的 4 位用来反映运算结果是否小于 0、大于 0、等于 0 或者有溢出;浮点运算指令后带‘.’,说明除了进行普通运算外,还需要根据异常状态对CR寄存器的CR1 域赋值;比较指令可以给CR寄存器的任意域赋值,跳转指令需要根据指令中指定的CR寄存器域进行跳转方向的判断.

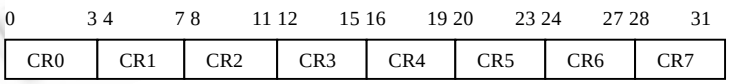



Fig.2 CR Register in PowerPC

图 2 PowerPC 的 CR 寄存器

如果二进制翻译的源系统有这类特殊寄存器,而目标系统没有,那么在翻译的过程中,就要考虑寄存器映射的问题.

最简单的寄存器映射方法是直接映射.直接映射的含义是把源系统上的特殊寄存器映射到目标系统的指定寄存器.以源系统是 PowerPC、目标系统是 Alpha 为例,可以将 PowerPC 上的 CR 寄存器映射到 Alpha 上的通用寄存器 Reg5.在该种映射方式下,设置 CR0 域时必须首先清空 Reg5 的最左边 4 位(对应到 CR0 域),然后再根据指令目标寄存器的结果设置条件值,最后将条件值拼入 Reg5;使用 CR0 域的时候,从 Reg5 中进行抽取.

图 3 是在直接映射方式下,PowerPC 上的指令序列翻译为 Alpha 上的指令序列的示意图.图 3 左部是 PowerPC 上的代码段,其中第 1 条 cmpwi 指令的含义是比较 r3 寄存器和立即数 0 的关系,如果 r3 小于 0,将条件值设为 8;如果 r3 大于 0,将条件值设为 4,如果 r3 等于 0,将条件值设为 2;条件值的最低位设置为与 XER 寄存器的 SO 位相同;将条件值送到 CR 寄存器的 CR0 域.第 2 条指令 addi 的含义是 r1 寄存器的值加立即数 8,放入寄存器 r29.第 3 条指令 bs 的含义是 CR0 域的的第 2 位如果为 1,则跳转到 0x1000dd48 处执行,否则,执行下一条指令.

图 3 右部是 Alpha 上的对应翻译序列,寄存器的映射关系为 PowerPC 的 r3 寄存器映射到 Alpha 的$0 寄存器,PowerPC 的 CR 寄存器映射到 Alpha 的$5 寄存器,PowerPC 的 r1 寄存器映射到 Alpha 的$7 寄存器,PowerPC 的 r29 寄存器映射到 Alpha 的$8 寄存器,PowerPC 的 XER 寄存器映射到 Alpha 的$4 寄存器.本次翻译实现了 PowerPC 指令的完整语义,左部的 cmpwi 指令被翻译为 Alpha 的 14 条指令,进行了条件值的设置和寄存器的拼接赋值;addi 指令比较简单,被翻译为 1 条指令;分支指令被翻译为 3 条指令,进行了寄存器特定位的抽取、比较.由图 3 可以看到,3 条 PowerPC 指令在采用寄存器直接映射的方式下,会被翻译成 18 条指令,虽然只使用了 5 个寄存器进行映射,但是在条件值的设置、拼接和寄存器特定位的抽取、比较中,需要引入 3 个临时寄存器存放中间值,所以不只代码膨胀率高,寄存器需求也较大.在频繁设置、使用 CR0 域的情况下,该种映射方式的效率相当差.

```
01. pc=0x1000dccc    cmpwi cr0,r3,0
02. pc=0x1000dcd0    addi r29,r1,8
03. pc=0x1000dcd4    bs 2,0x1000dd48
```

Code block on PowerPC

```
01.    sll $0,32,$1
02.    sra $1,32,$1
03.    cmplt $1,0,$2
04.    lda $3,4($31)
05.    cmovlbs $2,8,$3
06.    cmpeq $1,0,$2
07.    cmovlbs $2,2,$3
08.    srl $4,31,$2
09.    and $2,1,$2
10.    bis $2,$3,$3
11.    sll $5,36,$2
12.    srl $2,36,$2
13.    sll $3,28,$3
14.    bis $2,$3,$5
15.    lda $8,8($7)
16.    srl $5,28,$2
17.    and $2,2,$1
18.    bne $1,lab1
```

Code block on Alpha after translation

Fig.3 PowerPC→Alpha translation Example1 (direct mapping)

图 3 PowerPC→Alpha 翻译示例 1(直接映射)

## 2 寄存器分段映射

分段映射是可以提高程序执行效率的寄存器映射方法.分段映射的思想是把寄存器中的一位或者几位抽取出来,单独映射到一个寄存器.该种映射方式的充分必要条件是抽取出来的特殊位使用频繁,使用时可以独立于寄存器的其他位,而且该寄存器作为整体使用的时候比较少.

因为 PowerPC 的 CR 寄存器的 CR0 域、CR1 域是许多指令都要访问的区域,使用频繁,而且使用时,绝大多数情况下与 CR 寄存器的其他位没有任何关系,所以可以将 CR 寄存器进行分段映射.映射方法是 CR0 域单独映射到寄存器 Reg1,CR1 域单独映射到寄存器 Reg2,CR 寄存器作为整体再映射到寄存器 Reg3.这样,在只使用 CR0 域、CR1 域的场合,可以直接使用寄存器 Reg1,Reg2,而不需要进行寄存器的移位和拼接;在需要使用 CR 寄存器整体的时候,再将 Reg1,Reg2 的值拼接到 Reg3,虽然该种情况下翻译的比较复杂,但是根据 spec2000 目标码的分析情况,在程序的核心段,对整个 CR 寄存器进行操作的指令是不多的.

图 4 是采用分段映射方法处理特殊寄存器的示例,左部的 PowerPC 指令序列和图 3 一样.从图 4 可以看出,采用分段映射的有两个寄存器,CR 寄存器的 CR0 域映射到$5,XER 寄存器的 SO 位映射到$4,所以条件值形成时直接使用了$4 寄存器,分支指令直接使用了$5 寄存器.从图 3、图 4 的对比可以看出,采用分段寄存器映射方法,不需要进行寄存器的移位、拼接和抽取,可以明显减少代码膨胀,提高翻译代码的执行效率.但是,因为要进行条件值的设置,所以临时寄存器的需求并没有减少.

```
01. pc=0x1000dccc      cmpwi cr0,r3,0
02. pc=0x1000dcd0      addi r29,r1,8
03. pc=0x1000dcd4      bs 2,0x1000dd48
```

Code block on PowerPC

```
01.     sll $0,32,$1
02.     sra $1,32,$1
03.     cmplt $1,0,$2
04.     lda $3,4($31)
05.     cmovlbs $2,8,$3
06.     cmpeq $1,0,$2
07.     cmovlbs $2,2,$3
08      bis $4,$3,$5
09.     lda $7,8($6)
10.     and $5,2,$2
11.     bne $2,lab1
```

Code block on Alpha after translation

Fig.4 PowerPC→Alpha translation Example2 (segment mapping)

图 4 PowerPC→Alpha 翻译示例 2(分段映射)

## 3 寄存器功能剪裁

在二进制翻译中采用分段映射虽然可以提高翻译程序的执行效率,但是代码膨胀率仍然是比较高的.拿图 4 的示例来说,3 条代码翻译为 11 条代码,代码膨胀了接近 3 倍.如此高的代码膨胀率,必然会降低指令 cache 的命中率,影响翻译后课题的效率.为此,我们设计了特殊寄存器功能剪裁优化来进一步减小二进制翻译中的代码膨胀.

通过分析 PowerPC 的指令系统和 PowerPC 上生成的 SPEC2000 的可执行代码,我们发现特殊寄存器的设置和使用基本上是一一匹配的.以 CR 寄存器为例,设置 CR 寄存器的指令后不远必然有使用 CR 寄存器的指令.不过,设置和使用从完整性来说,并不是完全匹配,通常使用 CR 寄存器确定转换方向的指令只使用“小于”、“大于”、“等于”位,“溢出”位使用的非常少.

在二进制翻译的过程中,对于源系统的指令,在目标系统上通常是完全按照该指令的语义进行翻译的.所以特殊寄存器的位设置就成为代码膨胀的主要原因.如果不关心指令的完整语义,而是从指令功能上考虑翻译问题,那么代码膨胀应该就可以得到缓解,因为不同的处理器架构虽然有不同的指令系统,但是指令系统的完备性应该是不容置疑的.一个指令系统中的一种指令功能,在另外一个指令集结构中应该能够找到对应的类似功能.

这就是特殊寄存器功能剪裁优化的基础.

特殊寄存器功能剪裁优化的基本思想是根据特殊寄存器的使用情况,在二进制翻译的过程中,不进行指令语义的完整翻译,而是对指令的语义进行剪裁,进行指令功能的对等翻译.该项优化是在寄存器分段映射的基础上进行的,专门针对 CR 寄存器的 CR0 域.具体实现准则为:

**准则 1**. 对于设置 CR 寄存器 CR0 域的指令,翻译时不按照指令的原有语义生成条件值,而是直接将指令运行结果放入 CR0 域的映射寄存器.

具体来说,翻译 PowerPC 指令集中带'.'的算术运算指令和逻辑运算指令时,直接将目标寄存器的结果拷贝入 CR0 域的映射寄存器;翻译比较类指令时,如果使用 CR 寄存器的 CR0 域存放比较结果,则不进行比较运算,而是将比较指令的两个源操作数的差放入 CR0 域的映射寄存器.

**准则 2**. 对于使用 CR 寄存器的 CR0 域的条件分支指令,翻译时直接使用 CR0 域的映射寄存器的值作为判断条件,条件分支的形式可能要进行变形.

PowerPC 上使用条件值进行分支走向判断的功能实际上被替换为 Alpha 上的使用结果寄存器的值进行分支走向判断.对于判断 CR0 域的哪一位为 1 则跳转的条件分支,分支的形式可能要变成小于跳转、大于跳转、或者等于跳转.对于判断 CR0 域的哪一位为 0 则跳转的条件分支,分支的形式可能要变成小于等于跳转、大于等于跳转或者不等跳转.

**准则 3**. 对于使用 CR 寄存器的 CR0 域的其他指令,翻译时,首先要进行域值恢复,然后再进行 CR0 域的使用.

从以上 3 个准则可以看出,特殊寄存器功能剪裁的思想是在定值点剪裁 CR0 域的条件值设置,在使用点根据使用指令的情况,或者进行域值恢复,或者直接使用结果寄存器的值进行跳转方向判断.在 PowerPC 中,使用 CR 寄存器有 3 类指令,一类是条件分支指令,一类是 CR 寄存器内部拷贝指令,还有一类是将 CR 寄存器的值移出到通用寄存器的寄存器传送指令.在这 3 类指令中,条件分支指令是使用最多的指令,也是特殊寄存器功能剪裁优化的主要对象,能够直接受惠于寄存器剪裁优化.其他两类指令因为要进行域值恢复,所以翻译时,可能会增加指令膨胀.

图 5 是特殊寄存器功能剪裁的一个示例.左部 PowerPC 的代码序列和图 3、图 4 是一样的,右部是使用特殊寄存器功能剪裁后,翻译为 Alpha 上的代码序列.从图中可以看到,比较指令 cmpwi 被翻译为 3 条指令,未进行条件值的设置,只是将比较指令的两个源操作数的差放入了 CR0 域的映射寄存器$5,条件分支指令直接使用$5 确定分支走向,不过分支形式进行了改变,从图 4 的 bne(不等于 0 则跳转)变为 beq(等于 0 则跳转).整个代码段从 3 条指令变为 5 条指令,代码膨胀控制在一倍以内,而且因为不进行条件值的设置,所以临时寄存器的需求较少,减轻寄存器使用压力.

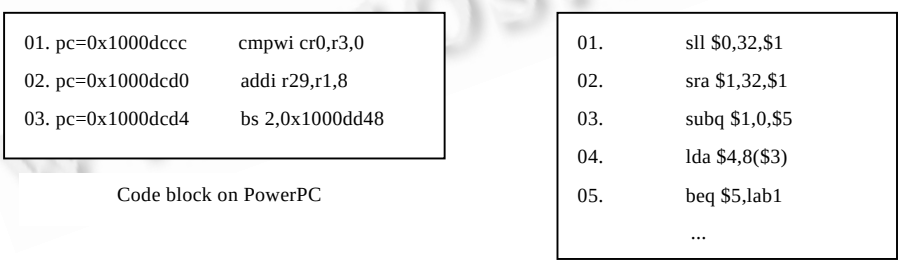



Fig.5 PowerPC→Alpha translation Example3 (function cutting out optimization)

图 5 PowerPC→Alpha 翻译示例 3(功能剪裁优化)

## 4 实验和测试

我们以 trend 系统为平台,对本文提出的寄存器分段映射和寄存器功能剪裁进行了实验和测试.

trend 系统是我们研制的一个动态二进制翻译系统,可以实现 PowerPC 上基于 Linux 的可执行程序在 Alpha 上的正确运行.trend 进行动态二进制翻译的基本策略是首先查找基本块,以基本块为单位进行指令的对等翻译,同时在基本块头添加语句进行基本块执行次数的 profiling,然后根据是否到达运行次数阈值确定程序运行的热路径,对热路径进行翻译、优化和执行.在 trend 系统中,我们分别采用直接映射方法、分段映射方法和寄存器功能剪裁进行了寄存器映射的实验.

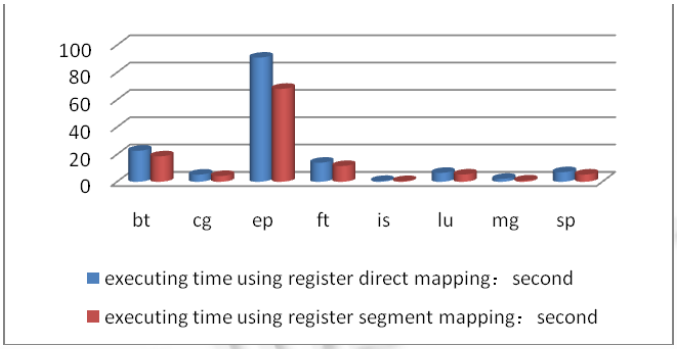



Fig.6 Performance comparison between register direct mapping and register segment mapping

图 6 使用寄存器直接映射和寄存器分段映射的性能对比

图 6 是直接映射和分段映射的效果对比,使用的测试用例是 NPB-serial 测试包的 8 道课题,使用的测试规模是 S 规模.8 道课题在 PowerPC 上使用 gcc 编译器 O3 选项静态链接生成后,在 466Mhz 的 Alpha 21264 上进行了测试.由图 6 可以看出,采用寄存器分段映射后,因为简化了常用指令的翻译,减少了代码膨胀,所以程序的运行时间明显减少.

图 7 是寄存器分段映射和寄存器功能剪裁的效果对比.测试用例是 SPEC2000 中的 10 道应用课题,测试规模是 test 规模.10 道课题在 PowerPC 上使用 gcc 编译器 O3 选项静态链接生成后,在 466Mhz 的 Alpha 21264 上进行了测试.图 7 中的“优化前”指的是采用寄存器分段映射,“优化后”指的是采用寄存器功能剪裁.从图中可以看出,使用寄存器功能剪裁进一步降低代码膨胀率后,使用动态二进制翻译运行程序的运行效率均有所提高,其中 186.crafty 和 253.perlbmk 课题的效果相当明显.通过反汇编查看这两道题的代码特征,发现存在大量的比较和条件判断指令,许多基本块和图 5 左侧的示例相似.

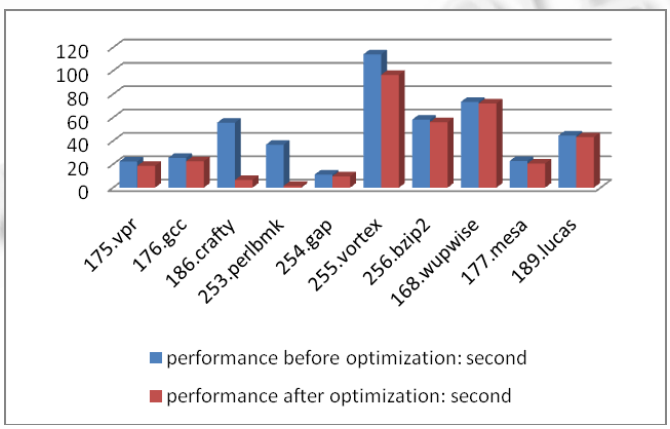



Fig.7 Performance comparison between register segment mapping and register function cutting out

图 7 使用寄存器分段映射和寄存器功能剪裁的性能对比

## 5 结束语

如何进行异构机之间可执行程序的高效移植是二进制翻译面对的难点问题,本文从寄存器映射的角度分析了这个问题,并针对源系统的具体特性提出了寄存器剪裁优化的设想.从本文可以看出,二进制翻译中的有效优化是建立在深入分析源、目标结构的具体特性的基础上的,结合具体架构的具体特点进行针对性优化是提高翻译后程序效率的一种有效手段.

**致谢** 在此,我们向对本文的工作给予支持和建议的同行表示感谢.

**References**:

[1] Bala V, Duesterwald E, Banerjia S. Transparent dynamic optimization: The design and implementation of dynamo. HPL-1999-78, HP Laboratories Cambridge, 1999.

[2] Klaiber A. The technology behind crusoe™ processors. Transmeta Corporation, 2000.

[3] Ebcioglu K, Altman ER. DAISY: Dynamic compilation for 100% architectural compatibility. Computer Science, RC 20538 (08/05/96).

[4] Hookway RJ, Herdeg MA. DIGITAL FX!32: Combining emulation and binary translation. Digital Technical Journal, 1997,9(1):3–12.

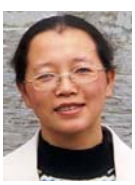
[5] Cifuentes C, Lewis B, Ung D. Walkabout—A retargetable dynamic binary translation framework. Technical Report, SMLI TR-2002-106, 2002.

[6] Robinson A. Why dynamic translation? Transitive Technologies, 2001.

[7] Cifuentes C, Van Emmerik M. UQBT: Adaptable binary translation at low cost. Computer, 2000,33(3):60–66.

[8] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture. Order Number: 253665-021, 2006.

[9] Power ISA™ Version 2.04. IBM, 2007.



**文延华**(1972－),女,湖南东安人,高级工程师,主要研究领域为二进制翻译,并行编译.

**唐大国**(1974－),男,工程师,主要研究领域为二进制翻译,并行编译.

**漆锋滨**(1966－),男,高级工程师,主要研究领域为并行体系结构,编译技术.