第 35 卷 第 9 期 计 算 机 工 程 2009 年 5 月
Vol.35 No.9 Computer Engineering May 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)09—0283—03 文献标识码：A 中图分类号：TP311

动态二进制翻译中的中间表示

姜玲燕，梁阿磊，管海兵
(上海交通大学软件学院，上海 200240)

摘 要：在二进制翻译中采用中间表示，可以适当隔离不同机器平台的特点，便于二进制翻译系统的移植。提出一种 VINST 中间表示方法，介绍其指令集与特点，运用 SSA 形式化和冗余指令删除等方法对 VINST 进行初步优化。优化前后的性能比较结果表明，相对简单高效的方法可以弥补优化的开销，提高系统性能。

关键词：动态二进制翻译；中间表示；二进制翻译器 CrossBit

Intermediate Representation in Dynamic Binary Translation

JIANG Ling-yan, LIANG A-lei, GUAN Hai-bing
(School of Software, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】 In binary translation, the use of intermediate representation can appropriate segregate different machine platforms, and make binary translation system more portable. This paper presents an intermediate representation called VINST, introduces the instruction set and its characteristics. It discusses some optimization methods including SSA formalization algorithm and redundancy elimination algorithm. Through comparison between before and after the optimization, it shows that a simple and efficient method can cover the cost of optimization itself and improve system performance.

【Key words】 dynamic binary translation; intermediate representation; binary translator CrossBit

1 概述

动态二进制翻译是指在运行时将一种指令集体系结构的二进制代码实时翻译成另一种指令集体系结构的指令，并立即执行翻译后的代码。从 20 世纪 90 年代开始，动态二进制翻译技术得到了广泛的研究。FX!32 技术使主流的 X86/ Windows 应用程序能够运行在 Alpha/ Windows 平台上。Aries 技术将快速解释和动态翻译相结合，使 PA-RISC 应用程序在运行 HP-UX 操作系统的 IA-64 机器上执行。多源多目标的 UQDBT 技术仅通过重写描述文件就可完成到新机器的移植。然而，大多数二进制翻译器都是单源单目标的(一种体系结构到一种体系结构)的。笔者开发的动态二进制翻译器 CrossBit[1] 具有可重定向性和可扩展性，可以支持多种源机器和多种宿主机之间的翻译，且可以添加新的源机器和宿主机。

VINST 中间表示的引入是为了支持 CrossBit 的可重定向。VINST 可适当隔离不同机器平台，方便系统的设计与实现，便于二进制翻译系统的移植，无论是源机器改变，还是目标机器改变，只须相应地调整前端解码器或后端编码器即可实现一个适用于新机器的二进制翻译系统。CrossBit 作为一个动态二进制翻译系统，采用 VINST 会影响程序的性能，因此，VINST 的设计必须尽可能规则与精简。

UQBT[2]使用 2 种中间表示，机器相关的寄存器传输列表(Register Transfer Lists, RTLs)和与机器无关的高层寄存器传输语言(Higher-level Register Transfer Language, HRTL)。

Dynamite[3]在核心部分采用一种中间表示。核心在中间表示上进行的优化包括识别基本块和基本块组、识别热点、死代码删除、冗余指令删除等。

Dynamo[4]是一个动态优化器，热路径(hot trace)存入缓存之前，首先被转换成一种中间表示，称为“fragment IR”。然后优化器对中间表示采用了一些经典的优化技术，如分支处理、冗余代码删除等。

Valgrind[5]也采用了一种中间表示，它是类 RISC 指令：具有 load/store 结构，且是平台无关的。Valgrind 对中间表示采用的主要优化技术包括冗余的 get 和 put 指令删除、常量传播、死代码删除等。

2 VINST中间表示分析

中间指令的设计直接决定了二进制翻译系统的运行时性能与生成代码的质量。中间指令需要尽可能保持简单以简化中后端对代码块的操作，尽可能保持源程序里每条指令的语义，并且尽可能多地支持多种指令集的不同特性。

VINST 的设计参考了 LLVA[6]和 VCODE[7]指令集，遵循计算机系统设计中的一条基本原则：使执行频繁的部分保持高效，使其他部分保持正确。因此，它只包含了各种指令集中最常用的二十余条指令，剩下相对不常用的简单指令用多种中间指令模拟，复杂指令则利用 CALL 中间指令，让高级语言的函数来模拟。

VINST 接近机器语言，是一种低层次的类 RISC 指令集，具有以下特性：

(1)具有无穷多个 32 位寄存器，称虚拟寄存器，记作 *vn*，其中，*v*0 永远代表零值。

(2)Load-Store 风格体系结构，即只有 Load 与 Store 2 种

基金项目：国家“863”计划基金资助项目(2006AA01Z169)；国家“973”计划前期研究专项基金资助项目(2007CB316506)；国家自然科学基金资助项目(60773093)

作者简介：姜玲燕(1982－)，女，硕士，主研方向：虚拟机，二进制翻译，中间语言；梁阿磊、管海兵，副教授、博士

收稿日期：2008-09-21 **E-mail**：lingyan@sjtu.edu.cn

指令可以访存。这种简化的访存方式使所有的访问内存操作显式化，便于进一步优化。

(3)唯一的寻址模式：偏移寻址(displacement)。所有的内存地址都可以记作：*vn*(*disp*)，内存地址等于寄存器 *vn* 的值加上偏移量 *disp*。偏移寻址可以衍生表达其他常用的寻址模式：*vn* 取 *v*0 即等于立即数寻址的情况，*disp* 取 0 即等于寄存器寻址的情况。

(4)VINST 指令只存在于内存中。

(5)每条 VINST 都是基本的，其功能不能用其他 VINST 指令完成，这种细颗粒度的指令有利于进行分析转换。

VINST 包含了与主流机器指令集(包括 RISC 与 CISC)相匹配的 6 类基本指令：运算，控制转移，数据移动，内存访问，寄存器状态映射，特殊指令。VINST 的指令集(不包括浮点指令)如下：

```
GET   s, v
PUT   v, s
LD    (v, imm), size, v
ST    v, size, (v, imm)
MOV   v1, v2
LI    imm, v
ADD   v1, v2, v3
SUB   v1, v2, v3
MUL   v1, v2, v3, v4
MULU  v1, v2, v3, v4
DIV   v1, v2, v3, v4
DIVU  v1, v2, v3, v4
AND   v1, v2, v3
OR    v1, v2, v3
XOR   v1, v2, v3
NOT   v1, v2
SLL   v1, v2, v3
SRL   v1, v2, v3
SRA   v1, v2, v3
SEXT  v1, size, v2
ZEXT  v1, size, v2
CMP   cc, v1, v2, v3
JMP   (v, disp)
BRANCH  cc, v1, v2, (v, disp)
HALT
SYSCALL
CALL  w1, w2, w3 …
```

2.1 VINST生成

本文给出一个简单的例子来说明 VINST 的生成方式。源指令是一段 PISA 指令，如下：

```
LW    s29, 0X37f76010(s31)
ADDIU   0X18, s29, s29
JR    s29
```

经过CrossBit前端解码后生成的VINST中间指令见表1。

表 1　PISA 指令生成的 VINST 指令

源指令	VINST 中间指令
LW　s29, 0X37f76010(s31)	(1)GET　s31, v1 (2)LD　(v1, 0X37f76010), WORD, v2 (3)PUT　v2, s29
ADDIU　0X18, s29, s29	(4)LI　0X18, v2 (5)GET　s29, v1 (6)ADD　v1, v2, v3 (7)PUT　v3, s29
JR　s29	(8)GET　s29, v1 (9)JMP　v1(0X0)

经过这一过程，3 条源指令被翻译成了 9 条中间指令。可见，VINST 的表示形式简单快速，但是效率却很低。因此，对 VINST 中间指令进行适当优化是非常必要的。

2.2 VINST优化

在上文的例子中，查看生成的 VINST 指令：对于每一条 PISA 指令，基本都是先从源寄存器 GET 操作数，进行处理，然后把结果 PUT 回去。由于指令之间存在依赖关系，因此生成了很多冗余的 GET/PUT 指令。比如第 8 条和第 5 条。基于这种情况，本文提出一种冗余指令删除算法。

2.2.1 SSA(Static Single Assignment)形式化

由于每条源指令翻译为中间指令时使用的虚拟寄存器是独立的，比如第 1 条指令中的 *v*1 和第 5 条指令中的 *v*1 是不相关的。为了方便数据流分析，须将 VINST 指令转换成 SSA 形式。主要算法如下：

维护一虚拟寄存器重命名表 *renameT*，表项为<寄存器原名，重命名>。遍历每一条 VINST 指令：

(1)获得该指令的所有虚拟寄存器个数，记为 *n*。

(2)对该指令中的 *n* 个寄存器分别进行处理：1)如果是定义寄存器，则对它进行重新编号，并将命名前后的对应关系<寄存器原名，重命名>放入重命名表 *renameT*；2)如果是使用寄存器，则查找重命名表 *renameT*，获得它的重命名，更新当前指令。

2.2.2 冗余指令删除

对 VINST 指令的分析，PUT/GET 指令是最重要的一类冗余指令，因此，本文提出一种冗余指令删除算法。

维护 3 张表：源寄存器与虚拟寄存器的映射关系表 *stateT*，表项为<源寄存器，虚拟寄存器>；虚拟寄存器重命名表 *renameT*2，表项为<寄存器原名，重命名>；PUT 指令暂存表 *pendT*，表项即<PUT 指令>。

遍历每一条 VINST 指令：

(1)判断该指令类型。

(2)如果该指令是 GET sreg, vreg 指令，则首先查询 *stateT* 表，判断 *sreg* 是否已经被映射。如果找到某个表项<*sreg, vregx*>，说明已被映射，更新 *renameT* 表项<*vreg, vregx*>，删除该指令；否则，把<*sreg, vreg*>添加到表 *stateT* 中。

(3)如果该指令是非 GET 指令，查询 *renameT* 表进行寄存器的重命名：1)如果该指令是 PUT *vreg, sreg* 指令，更新 *stateT* 表项<*sreg, vreg*>，说明 *sreg* 更新了，并将该指令加入 *pendT* 表，删除该指令。2)如果该指令是一条跳转指令，检查 *pendT* 表中是否有 PUT 指令，如果有，则在该跳转指令之前插入所有 PUT 指令。

SSA 形式化后的 VINST 和优化后的 VINST 如表 2 所示。

表 2　SSA 形式化后的 VINST 和优化后的 VINST

SSA 形式的 VINST	优化后的 VINST
(1)GET　s31, v1 (2)LD　(v1, 0X37f76010), WORD, v2 (3)PUT　v2, s29	(1)GET　s31, v1 (2)LD　(v1, 0X37f76010), WORD, v2
(4)LI　0X18, v3 (5)GET　s29, v4 (6)ADD　v3, v4, v5 (7)PUT　v5, s29	(4)LI　0X18, v3 (6)ADD　v3, v2, v5
(8)GET　s29, v6 (9)JMP　v6(0X0)	(7)PUT　v5, s29 (9)JMP　v5(0X0)

由此可见，VINST 经过优化后其指令数量减少，质量得到了提高。

3　实验数据分析

为检查优化算法的有效性，笔者在 CorssBit 上测试了程序包 Spec2000 中的部分 benchmark，程序的运行时间取自 5 次测试的算术平均值(用户时间和系统时间之和)，目标平台配置为 2.4 GHz Interl(R) Core(TM) 双核处理器、3.5 GB 内存，操作系统为 Linux。

优化前后的运行时间比较如图 1 所示(假设优化前为 1)。可以看出，对于大部分 benchmark，冗余指令删除优化都可以缩短系统的运行时间，带来 10%~40%的性能提高。

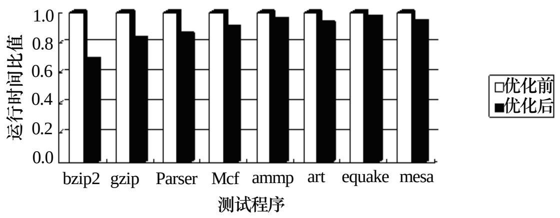

图 1　优化前后的运行时间比较

VINST 优化前后的代码量比较如图 2 所示(假设优化前为 1)。可以看出，对于大部分 benchmark，冗余指令删除优化都可以大大减少中间代码数量，甚至可以减少 30%~45%。

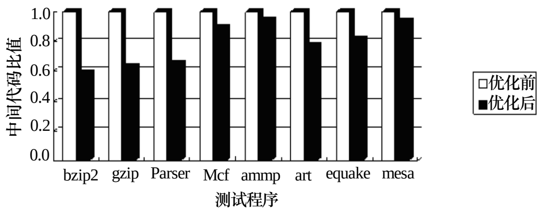

图 2　优化前后的代码量比较

综合图 1 和图 2 可知，程序运行时间的减少主要是因为 VINST 指令的减少，使得从 VINST 生成目标代码的时间缩短，目标代码数量相应减少，缩短了程序的运行时间。但是因为采用 SSA 算法和冗余删除算法需要 2 次遍历 VINST 指令进行处理，所以这部分消耗也不容忽视。此外，ammp, art, equake, mesa 是浮点程序，由于算法的限制，对其优化效果不是很理想，因此有待进一步研究。

4　结束语

本文介绍了 VINST 中间表示的设计思想及指令集，提出 SSA 形式化和冗余指令删除方法，在 CrossBit 上进行了测试，实验结果表明，冗余指令删除优化法提高了翻译器的性能，改进了生成代码的质量。下一步的研究方向是完善 VINST 指令集的设计，优化动态二进制翻译器。

参考文献

[1] 包云程. 构建基于动态二进制翻译技术的进程虚拟机[D]. 上海: 上海交通大学, 2007.

[2] Cifuentes C, Emmerik M V. UQBT: Adaptable Binary Translation at Low Cost[J]. IEEE Computer, 2000, 33(3): 60-66.

[3] Klaiber A. The Technology Behind Crusoe Processors[Z]. [S. l.]: Transmeta Corporation, 2000.

[4] Bala V, Duesterwald E, Banerjia S. Dynamo: A Transparent Dynamic Optimization System[C]//Proc. of the ACM Conf. on Programming Language Design and Implementation. Vancouver, British Columbia, Canada: [s. n.], 2000.

[5] Nethercote N, Seward J. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation[C]//Proceedings of the ACM Conference on Programming Language Design and Implementation. San Diego, California, USA: [s. n.], 2007.

[6] Adve V, Lattner C, Brukman M, et al. LLVA: A Low-level Virtual Instruction Set Architecture[C]//Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture. San Diego, California, USA: [s. n.], 2003.

[7] Engler D R. VCODE: A Retargetable, Extensible, Very Fast Dynamic Code Generation System[C]//Proc. of ACM Conf. on Programming Language Design and Implementation. New York, USA: [s. n.], 1996.

编辑　顾姣健

~~~~~~

（上接第 282 页）

6　结束语

PDA、智能手机等手持设备越来越成为人们移动办公和数字生活必不可少的组成部分，基于 GPS 定位为用户提供 LBS 服务的嵌入式 GIS 会是今后 GIS 系统的发展方向。本文将这类系统概括为“3S 系统”，基于世博会项目经验提出了典型 3S 系统的设计框架，并探讨了系统实现过程中涉及的关键技术，对类似系统的设计与实现有借鉴意义。今后 3S 系统的研究方向主要有:

(1)客户端软件实现完全跨平台，支持主流的 WinCE 和嵌入式 Linux 平台。

(2)根据系统具体特点对地图匹配、路径导航等关键算法进行优化。

(3)扩展服务器端软件功能，为用户提供更多人性化的 LBS 服务。

参考文献

[1] Sadoun B, Al-Bayari O. LBS and GIS Technology Combination and Applications[C]//Proc. of AICCSA’07. [S. l.]: IEEE Press, 2007.

[2] Chen Feixiang, Yang Chongjun, Yu Wenyang, et al. Research on Mobile GIS Based on LBS[C]//Proc. of IGARSS’05. [S. l.]: IEEE Press, 2005.

[3] Quddus M A, Qchieng W Y, Zhao Lin, et al. A General Map Matching Algorithm for Transport Telematics Applications[J]. GPS Solution Journal, 2003, 7(3): 157-167.

[4] Stentz A. The Focussed D* Algorithm for Real-time Re-planning[D]. Pittsburgh: Robotics Institute of Carnegie Mellon University, 1995.

编辑　张　帆
~~~~~~