航空科学技术
Aeronautical Science & Technology
Aug. 25 2020 Vol. 31 No.08 73-78

基于LLVM的多目标高性能动态二进制翻译框架

石磊[1,*]，田娜[2]，康烁[3]
1.中国空空导弹研究院，河南 洛阳 471000
2.北京迪捷数原科技有限公司，北京 100085
3.清华大学 信息技术研究院，北京 100084

摘　要：动态二进制翻译技术是构造高性能异构虚拟机的关键技术，而代码翻译的质量则是动态二进制翻译性能的关键。本文实现了一种基于LLVM动态二进制翻译框架，利用LLVM优化技术以及多目标编译的特点，实现了可以针对多个目标体系结构的高性能动态二进制翻译。基于开源Skyeye实现了这种翻译框架，并在两种目标体系结构ARM和PowerPC上验证了框架的可移植性和运行效率，与QEMU在ARM目标平台上做了性能对比，结果表明该模拟器比Qemu性能平均快10%以上。

关键词：动态二进制翻译；异构虚拟机；翻译性能；LLVM；多目标编译

中图分类号：TP314　　**文献标识码：**A　　DOI：10.19452/j.issn1007-5453.2020.08.012

测试性验证是指“为确定装备是否达到规定的测试性要求而进行的试验与评价工作”。当前绝大多数测试验证通过硬件来实现，受制于硬件条件，在硬件平台下基于目标码的异常、伤害性验证无法实现，软件测试的充分性无法保证。而基于目标码的虚拟测试验证则是建模仿真技术在测试性领域的一个具体应用[1]，已成为验证技术的重要发展方向之一，核心技术包括中央处理器（CPU）虚拟化、芯片虚拟化、接口虚拟化等，重点是CPU虚拟化，其实现目标机与主机（PC）的即时翻译，保证PC能够正确识别与运行目标机的目标码程序。

动态二进制翻译技术是一种即时翻译（JIT）技术，是把某种指令体系结构的二进制代码在运行过程中翻译成另外一种指令集体系结构的技术[2]。目标平台与PC体系结构相同，则为同构二进制翻译器，一般有两种用途：(1)实现热点执行路径优化，达到加速程序的目的，如Dynamo[3]；(2)对程序执行行为进行剖析，达到确定程序性能瓶颈的目的，如DynamoRIO，PIN，Valgrind等[4]。

当翻译目标指令集体系结构不同于PC指令集体系结构，则为异构二进制翻译器[5]。如谷歌的Android模拟器，使用Qemu二进制翻译技术，把ARM体系结构翻译成PC指令集体系结构（X86或者X86_64）[6]，其他还包括FX!32[7]、UQDBT[8]和Walkabout[9]等。可把一个特定硬件平台的应用移植到另外一个异构的硬件平台，或者提供一个硬件平台无关的虚拟运行环境[10]，或者利用多物理域仿真技术构建仿真运行环境[11]。

弹载软件大多数是嵌入式系统，若要实现目标码的解释与运行，需构建一个多目标的异构二进制翻译器，实现对多目标架构所有指令集的解释翻译。实践表明，由于实现过程的高复杂性、通用性差，需要大量的时间、人力和优化才有可能完成[12]。

针对上述问题，利用LLVM[13]构建一个多目标高性能二进制翻译器Dyncom，目标是利用LLVM的代码优化和

收稿日期：2020-04-01；退修日期：2020-05-28；录用日期：2020-06-25
基金项目：航空科学基金（2017ZD12013）
*通信作者：Tel.：15538868517　　E-mail：smxsuperboy@126.com
引用格式：Shi Lei, Tian Na, Kang Shuo. LLVM-based multiple targets high-performance dynamic binary translation framework [J]. Aeronautical Science & Technology, 2020, 31(08): 73-78. 石磊，田娜，康烁. 基于LLVM的多目标高性能动态二进制翻译框架[J]. 航空科学技术，2020，31(08)：73-78.

生成完成二进制翻译过程中不同目标架构代码的优化以及生成本地指令等较为复杂又影响执行性能的操作，实现多目标、高性能翻译的目的，生成高效的PC二进制代码。

为了提高动态翻译执行性能又降低实现的复杂性，Dyncom的代码优化基于LLVM中间语言实现，其是一种硬件无关语言，通过优化可用于所有不同目标指令集体系结构的翻译，大幅减少对不同目标指令架构的优化实现难度。主要有三个优化：带有一定阈值的混合执行、基于踪迹(trace)的超级块构造以及基于寄存器映射的冗余读写消除。

1 Dyncom：设计和实现

Dyncom以LLVM为基础，包含指令翻译和通用框架，前者把目标体系结构的指令集逐条翻译为LLVM的中间语言，后者实现中间语言到PC指令的翻译，总体体系结构如图1所示。

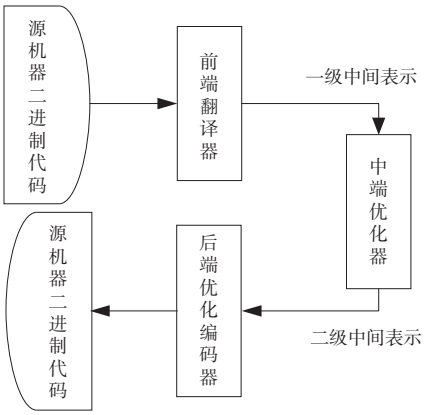

图1 总体体系结构

Fig.1 Overall architecture

1.1 LLVM模块

LLVM是一种类精简指令集架构的低级语言，加入一些高级语言特性，如类型系统、矢量表示和矢量运算等。借助类型，可对LLVM中间代码进行直接优化，而这在只有二进制信息的代码上是无法做到的，类型可分为基本类型(整型、浮点型、Void类型和标记类型等)和衍生类型(复合类型、Function类型和指针类型等)。Dyncom的中间语言为LLVM的虚拟指令集，目的是降低开源项目风险、降低开发成本以及易于各种虚拟机的开发。

LLVM中间语言指令有50条左右，包括二元、位、矢量、内存和终止操作指令等。例如，终止指令用于改变程序的执行流程，内存指令完成复杂内存操作，二元指令用于处理支持条件执行的架构指令集。

1.2 翻译过程

Dyncom指令翻译使用LLVM的虚拟指令集实现目标每一条指令翻译，是一个目标平台指令到LLVM指令间地址的一对多变换过程，包含三个部分：寄存器获取与条件执行检查、指令主体与条件标志更新以及寄存器写回。首先获取寄存器结构体指针，获取到相应通用寄存器/状态寄存器值，对于ARM架构，通过检查对应状态寄存器值决定是否执行指令，通过指令主体完成操作映射，通过条件标志更新完成通用寄存器值和PC更新，最后将其写回到对应结构体中。

ARM和SPARC(scalable processor ARChitecture)两条指令翻译过程示例见表1。

指令翻译以基本块(basic block，BB)为粒度，使用bb表示将被分析的BB块。第一行的SPARC指令不支持条件执

表1 Dyncom指令翻译

Table 1 Dyncom instruction translation

目标指令	中间代码
add %g2，%g4，%g3 (SPARC)	%0 = load i32 * %SPARC_g2 %1 = load i32 * %SPARC_g4 %2 = add i32 %0，%1 %3 = load i32 * %SPARC_pc %4 = add i32 %3，4 store i32 %2，i32 * %SPARC_g3 store i32 %4，i32 * %SPARC_pc
andsne r1，r1，r2 (ARM)	L_1： %CPSR_z_1 = load i32 * %CPSR_z %CMP_1 = icmp eq i32 %CPSR_z_1，0 br i1 %CMP_1，label %L_2，label %L_3 L_2： %1 = load i32 * %ARM_r1 %2 = load i32 * %ARM_r2 %3 = and i32 %1，%2 store i32 %3，i32 * %ARM_r1 %4 = load i32 * %ARM_r1 %5 = lshr i32 %4，31 store i32 %5，i32 * %CPSR_n %6 = load i32 * %ARM_r1 %CMP_2 = icmp eq i32 %6，0 %7 = select i1 %CMP_1，i32 1，i32 0 store i32 %7，i32 * %CPSR_z L_3： %8 = load i32 * %ARM_pc %9 = add i32 %8，4 store i32 %9，i32* %ARM_pc

行，对add指令的翻译按照一对多且不需要更新条件标志的步骤实现。第二行的ARM指令包含条件执行，在运行时对相应的标志进行判断，变换时，首先需要加载状态寄存器值，根据判断指令是否执行，其次生成两个bb块：L2和L3，L2包含and指令的翻译以及执行后Z和N标志位的更新，L3则是PC的更新，即无论and指令是否执行，PC都会更新。

指令翻译是一个中间无跳转指令的顺序指令流，当前端翻译到函数调用、跳转、返回指令时，即从上一个bb块结束后的第一条指令到该条指令为止划分为一个基本块，Dyncom则使用指令执行踪迹的方式将多个基本块组成一个包含控制流的超级块，其具体实现如图2所示。

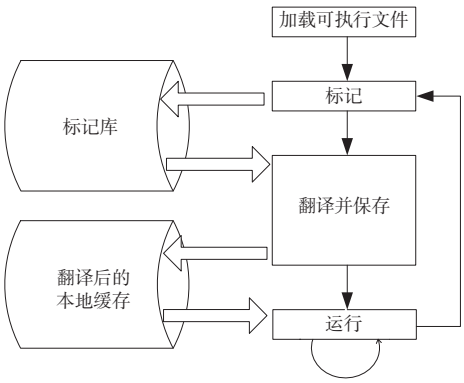

图2 翻译过程

Fig.2 Translating procedure

整个翻译过程被拆分成两个部分，标记(tag)和翻译(translate)，对代码进行两次扫描，第一遍是预先分析代码的性质，识别基本块和超级块，保存每一条目标代码指令的信息；第二遍对真正的扫描代码进行翻译。经过统计，扫描代码并打tag所占的时间是微乎其微的，会消耗一定量的内存，但为真正的翻译提供了必要的信息，简化了系统的复杂度，降低了开发难度和风险。翻译完成后，将每个基本块的地址作为哈希值记录到哈希表中，该哈希表被称为全局映射表，用于每个JIT执行退出返回到Dyncom时，查找下一个要执行的bb块所在的JIT入口地址。

1.3 执行过程

Dyncom采用混合执行方式，先使用解释模式执行目标指令的基本块，对执行过的目标指令块做统计。当发现某个基本块执行到一个阈值，认为该基本块是一个热点，然后使用Dyncom进行指令翻译，其翻译过程如上所述。指令继续解释执行直到翻译完成，并且执行到当前基本块结束，下一个跳转的基本块位于JIT中，此时开始动态执行，该方法可有效利用解释执行时间掩盖动态翻译的时间过载。

JIT内部执行则需要依靠本地映射表的存在，用来解析基本块的地址到本地代码。本地映射表在JIT中为一个dispatch基本块，其由一个大的switch指令构成，类似于C语言的switch控制流。每次进入JIT后，就会进入该基本块，dispatch根据PC的值确定在本地映射表中选择需要执行的基本块地址并跳转执行，基本块执行完毕后再次回到dispatch块，直到PC值不存在于本地映射表中，直接退出，将控制权交给Dyncom，通过全局映射表决定跳到下一个JIT执行还是进行翻译操作。至此，目标二进制程序不断被动态翻译并执行。

1.4 目标架构ARM和PowerPC

选择军工领域常用的ARM和PowerPC架构，用于验证动态二进制翻译框架对多目标架构的支持。通过Dyncom能够较为容易地把ARM指令或者PowerPC指令的应用程序翻译为中间语言，而从中间语言翻译到PC指令(X86或者X86_64)则由LLVM的后端完成。同理，如果需要支持更多目标体系结构，只需要把该目标体系结构的指令利用Dyncom翻译为LLVM的中间指令即可，本方案可以实现动态二进制翻译框架对多目标翻译的支持，具有良好的通用性。

2 运行时优化

2.1 JIT跳转优化

Dyncom动态翻译执行的粒度是逐块(block-by-block)方式，每执行完一个基本块，就回到基本块，通过本地映射表确定下一个跳转块，JIT中基本块没有控制流，而是构成了一个基本块同其他所有基本块组成的一个树状结构。LLVM对这种情况的优化程度有限，因此，本文对跳转类型分类，对能够在JIT内跳转的情况不再调度，减少本地映射表中存储地址的数量。

Dyncom的分支类型有两类：直接分支是在JIT编译时目的地址已知的分支，间接分支是在JIT编译时未知目的地址的分支。当翻译到一个分支时，会有以下情况：

(1) 直接分支

若目标分支不在JIT区域内，直接返回，由全局映射表决定执行步骤。

(2) 直接分支

若目标分支在JIT区域内，直接跳转到目标基本块，不再调度，从本地映射表中删除目标项。

(3) 间接分支

直接跳转到基本块处理。

经过跳转分类处理后JIT内部能够对分支跳转情况形成控制流，获取更多信息，分析发现，优化跳转后JIT代码体积明显变小，性能有一定提升。

2.2 全局寄存器映射

虽然LLVM中间表示（IR）提供了无穷虚拟寄存器，但其是静态单一赋值（SSA）形式，且目标架构的寄存器运行时状态值会一直变化，很难维护运行时状态，因此不能把每个目标架构的寄存器都匹配到一个LLVM寄存器。Dyncom通过指针将寄存器结构体传入JIT，每次获取寄存器值时，首先获取相应寄存器对应的地址，再通过加载指令获取到对应的值。分析发现，生成的IR中包含大量的地址计算，属于冗余指令。

使用LLVM的alloca指令实现JIT基本的寄存器的映射，用于在程序栈上分配空间给局部变量，一般用于处理函数参数。在JIT的entry基本块，首先获取所有寄存器对应的地址，其次使用alloca指令分配栈空间并将寄存器地址存入后，再次读取该栈空间的值，后续使用时，直接通过load指令读取该地址的值即可，该优化可减少每次获取寄存器值时需要的地址计算，且只在entry基本块中存储和加载。

2.3 基本块粒度冗余指令消除

经过全局映射优化处理后的JIT，虽然地址计算大大减少，但在基本块中，对于寄存器状态的处理过程一般为在基本块起始位置读入寄存器到内存的临时变量中，然后对其进行操作，在基本块运行结束之前再将对应临时变量的值写回到原有寄存器，而且指令与指令操作数之间原有的依赖导致通过LLVM原有的优化策略无法消除由于SSA形式产生冗余指令的情况。

本文是针对单一执行流的基本块进行冗余指令的优化，更大范围的交由LLVM处理。翻译过程中，为每个基本块建立一个LLVM虚拟寄存器到目标寄存器的映射表，每次加载寄存器之前，先扫描映射表，若表中没有，则加载，并存入到表中，若存在，则直接使用。写回寄存器之前，同样扫描映射表，若表中没有，则存入；否则，更新表中存在的值。当基本块翻译结束时，将表中所有已标记值写回到对应寄存器，同时清空映射表。该项优化减少了大量包含依赖关系的load/store指令，性能提升较大。

3 试验以及评估

为评估Dyncom的性能，进行对比试验，试验环境见表2。

表2 试验环境

Table 2 Experimental environment

硬件/软件	配 置
CPU	Intel（R）Core（TM）i7-2600K CPU @3.4GHz
内存	8G
操作系统	Ubuntu14.04
Skyeye	1.3.4-rc1/LLVM2.8
QEMU	2.0.0
Benchmark	EEMBC cjpeg1000

使用基于LLVM 2.8的Skyeye 1.3.4-rc1同QEMU 2.0.0进行性能比对测试，测试用例为EEMBC的cjpeg1000，运行结果以时间秒为单位，为了减少误差，样本运行100次，并取平均值。运行结果见表3，结果显示，优化后平均性能比QEMU高10%以上。

表3 运行时间

Table 3 Execution time

DBT	运行结果/s
Skyeye 1.3.4-rc1	13.584
QEMU 2.0.0	14.805

4 结论

本文首先阐述了动态二进制翻译的基本概念，说明Dyncom使用LLVM作为动态二进制翻译框架的原因。其次，分析Dyncom的实现框架和工作原理，详述翻译过程和执行过程，并在LLVM IR层次实现跳转优化和基本块粒度的冗余消除优化，提升了性能。最后，通过与QEMU进行对比，证明基于LLVM的Dyncom动态翻译的高性能，以及多目标翻译框架代码的可实现性。

相比于QEMU，Dyncom只关注目标体系架构到LLVM中间指令的翻译过程，中间语言到PC体系架构的指令翻译由LLVM自动完成。在最新的LLVM 10.0版本中，LLVM支持常用的ARM、PowerPC、X86、RiscV等不同的体系结构，几乎不用做任何移植工作，即可把目标体系架构翻译为LLVM支持的数十种后端体系结构，但QEMU支持新的PC体系结构则需要大量的手动翻译的编码实现，由此可见，Dyncom的可移植性远高于QEMU。

Dyncom在冗余消除粒度上仍然过小，JIT中仍然有很多冗余指令，由于其操作数跨基本块使用，导致无法对其进行删除，另外由于局部映射表对所有的基本块都进行了存

储，也限制了处理能力，本文后续将从这两点入手进行优化。

参考文献

[1] 张勇，邱静，刘冠军，等. 面向测试性虚拟验证的功能–故障–行为–测试–环境一体化模型[J]. 航空学报，2012，33（2）：273-286.
Zhang Yong, Qiu Jing, Liu Guanjun, et al. Integrated function-fault-behavior-test-environment model for virtual testability verfication[J]. Acta Aeronautica et Astroautica Sinica, 2012, 33（2）:273-286.（in Chinese）

[2] Tom S，Harry W，Björn F，et al. Efficient code generation in a region-based dynamic binary translator[C]// LCTES '14：Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages，Compilers and Tools for Embedded Systems. Edinburgh，United Kingdom，2014：3-12.

[3] Hsu C C，Liu P，Wu J J，et al. Improving dynamic binary optimization through early-exit guided code region formation [C]// Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York，United States，2013：23-32.

[4] Rodríguez R J，Artal J A，Merseguer J. Performance evaluation of dynamic binary instrumentation framework[J]. IEEE Latin America Transactions，2014，12（8）：1572-1580.

[5] Hsu C C，Liu P，Wang C M，et al. Lnq：Building high performance dynamic binary translators with existing compiler backends[C]// 2011 International Conference on Parallel Processing. Taipei，Taiwan，2011：226-234.

[6] Hsu C C，Hong D Y，Hsu W C，et al. A dynamic binary translation system in a client/server environment[J]. Journal of Systems Architecture，2015，61（7）：307-319.

[7] 李剑慧，马湘宁，朱传琪. 动态二进制翻译与优化技术研究[J]. 计算机研究与发展，2007，44(1)：161-168.
Li Jianhui, Ma Xiangning, Zhu Chuanqi. Dynamic binary translation and optimization[J]. Journal of Computer Research and Development, 2007, 44(1): 161-168.（in Chinese）

[8] Ung D，Cifuentes C. Dynamic binary translation using run-time feedbacks[J]. Science of Computer Programming，2006，60（2）：189-204.

[9] Cristina C，Brian L，David U. Walkabout：A retargetable dynamic binary translation framework[R]. Forth Workshop on Binary Translation. Virginia，United States，2002：1-30.

[10] 董卫宇，刘金鑫，戚旭衍，等. 基于热例程的动态二进制翻译优化[J]. 计算机科学，2016 (5)：27-41.
Dong Weiyu, Liu Jinxin, Qi Xuyan, et al. Hot-routine based optimization of dynamic binary translation[J]. Computer Science, 2016 (5): 27-41.（in Chinese）

[11] 聂同攀. 基于模型的机电系统多物理域仿真技术应用研究[J]. 航空科学技术，2017（7）：68-72.
Nie Tongpan. The simulation technology application research of model-based electromechanical systems muti-physical domain[J].Aeronautical Science & Technology,2017（7）：68-72.（in Chinese）

[12] 陈琐颢，郑重，沈立，等. 二进制翻译中代码生成的子图覆盖算法[J]. 计算机科学与探索，2011，5(7)：613-623.
Chen Xuhao, Zheng Zhong, Shen Li, et al. Subgraph covering for efficient code generation in binary translation[J]. Journal of Frontiers of Computer Science and Technology, 2011, 5(7): 613-623.（in Chinese）

[13] Carsten S，Florian M，Stephan F. LLBMC：a bounded model checker for LLVM’s intermediate representation[C]// International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Tallinn，Estonia，2012：542-544.

（责任编辑 陈东晓）

作者简介

石磊（1978– ）男，硕士，高级工程师。主要研究方向：弹载软件测试验证技术。
Tel：15538868517
E-mail：smxsuperboy@126.com

田娜（1983– ）女，学士，工程师。主要研究方向：嵌入式系统、处理器验证。
Tel：13501153049
E-mail：tianna@digiproto.com

康烁（1978– ）男，硕士，工程师。主要研究方向：处理器仿真及验证，软件安全。
Tel：13651119140
E-mail：ksh@skyeye.org

LLVM-Based Multiple Targets High-Performance Dynamic Binary Translation Framework

Shi Lei[1,*], Tian Na[2], Kang Shuo[3]

1. China Air to Air Missile Research Institute, Luoyang 471000, China

2. Beijing Digiproto Technology Co., Ltd., Beijing 100085, China

3. Research Institute of Information Technology, Tsinghua University, Beijing 100084, China

Abstract: Dynamic binary translation is a key technology of constructing high-performance heterogeneous virtual machine, while the quality of the code translation is the key of dynamic binary translation performance. An LLVM-based dynamic binary translation framework is implemented. Utilizing LLVM optimization technology and the features of multiple target compiling, the high performance dynamic binary translation for multiple target architectures is implemented. The dynamic binary translation framework is implemented based on open source software Skyeye. The portability and performance were verified on two target architectures ARM and PowerPC. Compared with Qemu on ARM target platform, experiment results show that the average performance is faster than QEMU by over 10%.

Key Words: dynamic binary translation; heterogeneous virtual machine; translation performance; LLVM; multiple targets translation

Received: 2020-04-01； **Revised:** 2020-05-28； **Accepted:** 2020-06-25

Foundation item: Aeronautical Science Foundation of China(2017ZD12013)

***Corresponding author.Tel.**：15538868517 **E-mail**：smxsupperboy@126.com