第 37 卷 第 20 期
Vol.37 No.20

计 算 机 工 程
Computer Engineering

2011 年 10 月
October 2011

·开发研究与设计技术· 文章编号：1000—3428(2011)20—0277—03 文献标识码：A 中图分类号：TP311.54

基于动态二进制翻译技术的仿真器研究

陈 乔，蒋烈辉，董卫宇，徐金龙，方 明

(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：以动态二进制仿真器 QEMU 为平台，分析动态二进制翻译技术在仿真器开发中的应用，研究 QEMU 的翻译机制、优化策略、关键技术，并对相关重要代码进行解析。对仿真 CPU 的性能进行测试，结合分阶段的测试结果，从中找出制约仿真 CPU 性能的关键阶段，为后续的优化工作提供参考依据。

关键词：动态二进制翻译；软件移植；中间指令；精确异常；自修改代码

Simulator Research Based on Dynamic Binary Translation Technology

CHEN Qiao, JIANG Lie-hui, DONG Wei-yu, XU Jin-long, FANG Ming

(College of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

【Abstract】 Using the dynamic binary translator QEMU as a research platform, this paper analyzes the dynamic binary translation technology used in the development of the simulator, studies in detail QEMU with the translation mechanism, optimization strategy, and key technologies and analyzes important related code. Simulation tests the performance of CPU, and combining phases test results, finds the restriction of the CPU performance simulation key stages, it provides reference for optimization.

【Key words】 dynamic binary translation; software migration; intermediate instructions; precise exception; self-modified code

DOI: 10.3969/j.issn.1000-3428.2011.20.094

1 概述

计算机发展的历程是软硬件相互矛盾、相互统一的过程。传统的计算机软件和硬件都依赖于特定的体系结构，不同的指令集体系结构的软件和硬件不能相互结合，降低了计算机系统的互操作性。动态二进制翻译技术解决了软件移植问题，不仅对软件重用有重大意义，而且还可以开阔处理器研发的思路，促进新的处理器的创新。本文以动态二进制仿真器 QEMU 为平台，研究了 QEMU 的翻译机制、优化策略和关键技术。

2 国内外研究现状

从 20 世纪 80 年代开始，人们主要使用二进制翻译来解决软件兼容问题。1992 年，DEC 公司就开发了 FX!32 仿真程序，它可以使 IA-32 应用程序在一个运行 Windows 操作系统的 Alpha 平台上透明执行。昆士兰大学 1999 年开发了一个开放的静态二进制翻译系统——UQBT，以及后续的动态二进制翻译系统——UQDBT。UQBT 是一个静态二进制翻译系统，具有多源多目标的特性，可以支持多种源机器和多种宿主机之间的翻译。QEMU(Quick EMUlator)是目前较为先进的支持多源平台的二进制翻译系统。由于需要进行跨平台仿真，QEMU 核心技术是多源到多目标(称可重定向)的动态二进制翻译。

国内目前已经有不少高校和研究所在从事二进制翻译技术的研究。中国航空计算技术研究所已经开发了 BTASUP 系统，能够在 PowerPC 处理器上实现对 1750 处理器的二进制可执行代码的透明执行。中科院成功开发了 Digital-Bridge 动态二进制翻译系统，可以在 MIPS 处理器上运行 x86 的程序。清华大学的 Skyeye 项目也在 ARM 的仿真器中添加了动态二进制翻译模块，从而明显地提高了仿真速度。

3 动态二进制翻译器 QEMU

QEMU 是一套多源多目的开源跨平台翻译器。QEMU 有 2 种主要运行模式：进程模式和系统模式[1]。在系统模式中 QEMU 能仿真整个硬件系统，包括中央处理器及其他周边设备。

3.1 QEMU 系统的基本结构

系统由控制核心、解释器、翻译器和翻译缓存等组成。控制器负责整个翻译过程的调度，解释器负责完成指令的匹配工作，翻译器主要完成源二进制指令流到中间指令和中间指令到目标二进制指令流的转换，最后翻译缓存负责存放翻译后的基本块。在运行时，控制核心会维护一个软件的目标机虚拟 CPU 状态，称为 env，它包括通用寄存器、段寄存器、标志位寄存器等。目标机的所有资源都通过 env 基址加上特定的偏移来访问。

3.2 QEMU 的翻译单元

动态二进制翻译器 QEMU 采用基本块作为翻译单元。所谓基本块是一段只有一个入口和一个出口的程序段，通常一个基本块包括 4 条~7 条指令[2]。选择基本块做翻译单元，相对于使用单条指令为单位的翻译，可以省去很多有关函数调用的操作，同时可以充分挖掘基本块内的指令并行性，给编译器提供了更多的优化空间。采用基本块作为翻译单元，动态二进制翻译器会把翻译和优化的结果保存起来，下次遇到

作者简介：陈 乔(1985－)，男，硕士研究生，主研方向：动态二进制翻译，计算机系统结构；蒋烈辉，教授、博士生导师；董卫宇，讲师、硕士；徐金龙、方 明，硕士研究生

收稿日期：2011-03-15 **E-mail：**www22@sina.com

相同代码段时，就可以直接执行预存起来的翻译后代码。

3.3 QEMU 的翻译与执行过程

动态二进制翻译器 QEMU 引入了中间指令 TCG(Tiny Code Generator)以协调多平台的语义差异，大体翻译框架如图 1 所示。

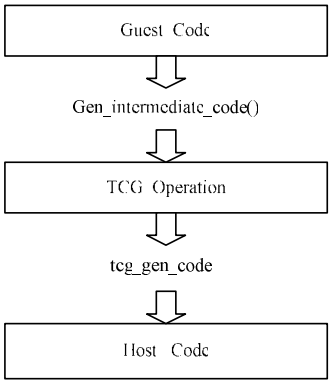

图 1 翻译框架

动态二进制翻译器 QEMU 在整个执行过程中可分为以下 3 个阶段[3]。

(1)查询。这个阶段查询目标代码块是否存在于代码缓存中。如果存在，则返回目标代码入口地址。

(2)翻译。如果所查询的代码块不在代码缓存中，则要对代码块进行翻译，完成从源平台二进制代码到目标二进制代码的翻译。

(3)执行。将翻译后的代码块在本地机上运行。

3.4 QEMU 的优化策略

所谓优化策略，就是确定何时对翻译后的基本块进行优化。显然，最简单的方法就是从不优化或者对每次遇到的没有翻译过的代码在翻译时都进行优化。然而，优化是有一定的开销的，有些代码在运行的过程中可能只会碰到一次。因此，系统需要找出值得进行优化的部分。现有的动态二进制翻译系统一般在基本块之间采用类似于解释执行的方式，即翻译一个基本块执行一个，当某个基本块被反复执行时，才单独对该基本块进一步优化，然后将优化后的结果存入 T-Cache 中，以备再次执行到该基本块时使用。QEMU 采用的就是这种处理方法。

4 动态二进制翻译器 QEMU 中的关键技术

4.1 中间指令

QEMU-0.10.0 引入了中间指令，从逻辑上隔离了源和目标体系结构。中间指令具有 RISC 特征，只包含那些各种源指令集中常用的操作，其余的源指令可能使用多条中间指令模拟，也可能使用 C 语言实现，并由中间指令 CALL 来调用。具体来说，QEMU 中间指令的特征如下：

(1)可使用无穷多个寄存器，称之为虚拟寄存器 VR，中间指令使用 VR 的索引来引用 VR。

(2)只有 Load/Store 2 类访存指令，且只有寄存器相对寻址一种内存寻址方式，可记做 disp(VRn)。

(3)其他中间指令的操作数均使用 VR 或立即数。

4.2 精确异常

精确异常是处理器的特性之一，对于采用流水线、多发射、乱序执行等指令级并行技术的处理器，当异常发生时，多条指令可能已经开始处于执行阶段，通常将引发异常的指令称为 exception victim。为了方便软件开发，处理器应该保证程序中位于 exception victim 之前的指令执行完毕，并且 exception victim 以及其后的指令不会对软件执行产生副作用。满足精确异常的处理器的指令执行具有原子性，并且从软件看来，指令是顺序执行的[4]。

QEMU 使用 setjmp/longjmp 支持精确异常，在虚拟处理器执行之前，使用 setjmp 设置还原点，以后发生异常时使用 longjmp 跳至还原点处理异常。通过 setjmp/longjmp，QEMU 实际将异常和中断的嵌套处理序列化，避免了整个翻译引擎的递归执行。控制代码翻译和执行的主控流程包含在 cpu-exec()函数中。

4.3 自修改代码

二进制代码可以在运行时修改自身，如果动态二进制翻译器不能发现这种情况，那么可能执行的是以前翻译的旧代码。对在操作系统之下的动态二进制翻译器，操作系统从磁盘数据调入内存页可以产生同样的问题，被修改的代码块必须被重新翻译，以保证与最新的代码同步。

QEMU 使用如图 2 所示的结构，管理 TB(Translation Block)与源内存页间的关系。从图 2 中可以看出，每个源内存页可能与多个 TB 关联，以后当某个物理页第 1 次被写入时，QEMU 使所有与该物理页关联的 TB 无效。

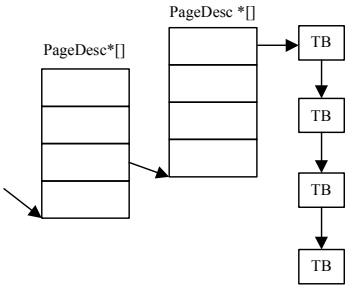

图 2 自修改代码管理示意图

5 动态二进制翻译器 QEMU 中的代码构建

5.1 主体代码的构建

主体代码的构建过程如下：

(1)TB 的结构在文件 translate-all.h 中进行定义。

(2)cpu-exec()函数设置在文件 cpu-exec.c 中，主要进行翻译块查询、代码块翻译和翻译块连接等工作。

(3)针对不同的体系结构，target-*/translate.c 主要完成从客户指令系统到中间指令的翻译工作。

(4)针对不同的体系结构，tcg-*/*主要完成从中间指令到主机指令系统的翻译工作。

(5)vl.c 文件中设置了系统仿真的主循环过程，是整个程序执行的主文件。

(6)hw/*主要进行硬件的仿真，包括视频、音频、网卡等。

5.2 主要运行阶段相关函数设置

5.2.1 查询阶段

在 QEMU 中采用 2 种寻址方式来进行代码 Cache 的索引，即虚拟地址索引和物理地址索引，查询过程函数设置如表 1 所示。

表 1 查询过程函数设置

执行函数	所在文件	备注
main_loop()	vl.c	这是程序执行的总循环函数
tcg_cpu_exec()	vl.c	
qemu_cpu_exec()	vl.c	
cpu-exec()	cpu-exec.c	
tb_find_fast()	cpu-exec.c	主要进行翻译块查询工作(虚拟地址进行查询)
tb_find_slow()	cpu-exec.c	主要进行翻译块查询工作(物理拟地址进行查询)

5.2.2　翻译阶段

在 QEMU 中引用了中间代码 TCG，完成从源代码到中间代码，再从中间代码到目标代码之间的转换。翻译阶段函数设置如表 2 所示。

表 2　翻译过程函数设置

执行函数	所在文件	备注
tb_gen_code()	cpu-exec.c	
cpu_gen_code()	Translate-all.c	
gen_intermmediate_code()	target-i386/Translate.c	源结构为 x86
gen_intermmediate_code_internal()	target-i386/Translate.c	
disas_inst()	target-i386/Translate.c	对 x86 指令进行解析生成中间指令
tcg_gen_code()	Tcg.c	
tcg_gen_code_common()	Tcg.c	

5.2.3　执行阶段过程中的上下文切换

当一个目标代码块被查询到或者由翻译模块生成时，翻译系统会执行一次上下文切换，切换时对相关的上下文信息进行保存，并由目标代码块入口开始执行，执行结束后再执行上下文恢复以继续翻译系统的执行。

在二进制翻译系统生成了目标代码块并放入缓存之后，翻译系统需从缓存系统中查询目标代码块的入口地址，并顺序完成从翻译系统到二进制代码执行时的寄存器上下文的保存、代码执行、上下文恢复整个过程。该过程由函数 tcg_target_qemu_prologue()完成。在 tcg/alpha/tcg-target.c 中实现函数 tcg_target_qemu_prologue()，该函数被 tcg.c 中的函数 tcg_context_init()所调用，生成 TB 的 prologue 和 epilogue。其中，函数 tcg_target_qemu_prologue()主要完成以下功能(宿主平台为 Alpha):

(1)在堆栈中保存 callee-saved 寄存器，这些寄存器可能被 TB 使用，因此，需事先保存。

(2)在堆栈中预留 TCG_STATIC_CALL_ARGS_SIZE 大小的内存，供 TB 调用 helper 函数时使用。

(3)跳转到 TB 执行，TB 的入口保存在$16 中。

(4)利用语句 tb_ret_addr=s->code_ptr 保存 TB 的返回地址。

(5)调整堆栈指针，清除堆栈中为 TB 调用 helper 函数预留的区域。

(6)恢复保存在堆栈中的 callee-saved 寄存器。

(7)生成 ret 指令。

其中，第(1)步~第(3)步是生成 prologue 代码，第(4)步~第(7)步是生成 epilogue 代码。

在整个保存和恢复的过程中把$26 压入堆栈中，保存返回地址，将$27、$28、$29 寄存器入栈，然后，保存 callee 寄存器内容，分别是$15、$9、$10、$11、$12、$13、$14，这些寄存器如果 Callee 修改，则必须对它们进行保存恢复。当 TB 运行结束后利用出栈指令将返回地址放入$26 中，返回源程序。

6　QEMU 性能评测与分析

采用测试集 SPEC2000 以 Ref 规模的输入对仿真 CPU 的性能进行评测。分别在利用 QENU 构建的系统虚拟机和通用 PC 本地运行，测试结果如表 3 所示。

表 3　仿真 CPU 性能数据　　s

程序	PC 本地机			PC 虚拟机		
	Reference Time	Base Runtime	Base Ratio	Reference Time	Base Runtime	Base Ratio
164.gzip	1 400	138.0	1 017*	1 400	1 314	107*
175.vpr	1 400	165.0	849*	1 400	1 892	74*
181.mcf	1 800	167.0	1 078*	1 800	593	303*
186.crafty	1 000	68.8	1 455*	1 000	2 001	50*
254.gap	1 100	67.9	1 620*	1 100	1 490	74.8*
256.bzip2	1 500	146.0	1 025*	1 500	1 261	119*

图 3 为测试结果的对比，从图中可以看出，QEMU 对性能的损耗是比较大的。经过统计性能基本维持在本地机性能的 10%~15%。

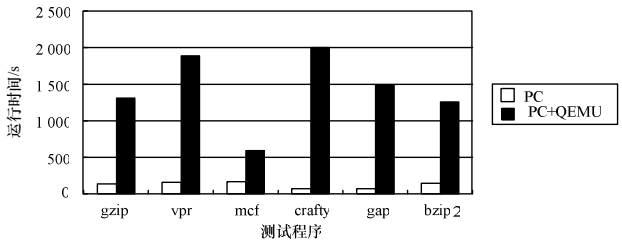

图 3　仿真 CPU 性能测试对比

分析整个翻译执行过程，对整个过程进行阶段划分，找出性能损耗最多的环节。如表 4 所示，运行程序显示各个阶段运行时间所占比例。

表 4　各阶段运行时间比例

运行阶段	所占比例/(%)
查询	24
翻译	48
执行	28

从表 4 中可以看出，在整个过程中翻译部分消耗的时间最多。QEMU 中引入了中间指令，一方面减少了在翻译过程中语义之间的差异，但是另一方面每次要经过 2 次翻译过程即源指令到中间指令，再从中间指令到目标指令。产生的代码膨胀率会很高，从而使仿真出的 CPU 的性能很低。翻译阶段成为了系统性能的瓶颈之一。

7　结束语

目前大部分动态二进制翻译系统没能达到用户的性能需求，通过实验，QEMU 的性能仅维持在本地机性能的 10%~15%，所以，动态二进制翻译器的优化就成了非常紧迫的任务。如果用户可以在宿主机上获得接近于源机器上的程序运行速度，动态二进制翻译技术将会得到更加广泛的应用。

参考文献

[1] Edgar E. Debugging and Profiling Embedded Linux/CRIS Systems with QEMU[EB/OL]. (2009-10-21). http://tree.celinuxforum. org/CelfPubWiki/ ELC2009Presentations?action=AttachFile&do= get&target=elc2009-qemu-cris.pdf.

[2] Fabrice B. QEMU: The Open Source Processor Emulator[EB/OL]. (2010-03-31). http://fabrice.Bellard. Free.fr/qemu/zbout.html.

[3] Chad D, Kersey. QEMU Internals[EB/OL]. (2009-10-21). http://lugatgt.org/content/qemu_internals/downloads/slides.pdf.RInternals.pdf.

[4] 张　激, 李宁波. 基于二进制翻译的仿真器关键技术研究[J]. 计算机工程, 2010, 36(16): 246-248.

编辑　索书志