·体系结构与软件技术· 文章编号: 1000-3428(2019)05-0072-05 文献标志码: A 中图分类号: TP391

面向国产平台的二进制翻译库函数替换优化处理机制

张家豪,单 征,岳 峰,傅立国,王 军

(数学工程与先进计算国家重点实验室,郑州 450001)

摘 要: 在二进制翻译中,库函数本地化处理是指不直接翻译源文件中的库函数而是使用本地目标机的库函数进行替换,以此提高翻译性能。针对国产平台二进制翻译中库函数处理翻译模式的相关特性,提出一种库函数解析处理机制,借助 helper 函数实现库函数替换操作。通过修改基本块划分规则,将库函数替换部分翻译的基本块进行合并,构建超级块,减少源程序中基本块划分数量与程序执行跳转次数,在保证翻译正确性基础上有效提高翻译效率。在动态二进制翻译器 QEMU 上进行的实验结果表明,与未优化的库函数处理翻译方式相比,翻译后程序加速比平均提升 9%,有效提高了翻译效率。

关键词: 二进制翻译; 库函数解析; 基本块合并; QEMU 翻译器; 国产平台

中文引用格式: 张家豪,单征,岳峰,等. 面向国产平台的二进制翻译库函数替换优化处理机制[J]. 计算机工程,2019,45(5):72-76,83.

英文引用格式: ZHANG Jiahao, SHAN Zheng, YUE Feng, et al. Binary translation library function replacement optimization processing mechanism for domestic platform[J]. Computer Engineering,2019,45(5):72-76,83.

Binary Translation Library Function Replacement Optimization Processing Mechanism for Domestic Platform

ZHANG Jiahao,SHAN Zheng,YUE Feng,FU Liguo,WANG Jun

(State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China)

【Abstract】 In binary translation,library functions localization processings do not directly translate library functions in source files but use library functions of local target machines to improve translation performance. Aiming at the related characteristics of library function processing translation mode in binary translation of domestic platform,a library function analysis processing mechanism is proposed. The library function replacement operation is realized by the helper function. By modifying the basic block partitioning rules,the library function replaces the partially translated basic blocks to merge, constructs the super block,reduces the number of basic block partitions and program execution jumps in the source program,and effectively improves the translation efficiency on the basis of ensuring the translation correctness. Experimental results based on the dynamic binary translator QEMU show that compared with the unoptimized library function processing translation method,the post-translation program acceleration ratio is increased by 9%,which verifies the effectiveness of the mechanism.

【Key words】 binary translation; library function analysis; basic block merging; QEMU translator; domestic platform

DOI: 10.19678/j.issn.1000-3428.0050157

0 概述

二进制翻译技术是解决软件移植问题的重要手段,在遗产代码移植、程序优化、系统安全等方面都有着重要的意义[1]。二进制翻译系统的出现,对国产新平台的推广具有重要意义,它使得应用软件可以通过二进制翻译系统直接在新体系结构平台上运行,减少了对专用硬件结构的依赖。二进制翻译技术是解决国产 CPU 平台对主流软件系统兼容的重要选择之一,能够给国产 CPU 体系提供创新发展的机遇。近年来,国内外相关领域学者针对此技术进行了深入的研究。

库函数本地化处理是二进制翻译中提高翻译效率的热点技术,其核心思想是在二进制翻译阶段将源二进制文件中的库函数调用按目标机的传参规则准备好参数,然后调用目标机的库函数,即无需翻译

基金项目: 国家自然科学基金(61472447); 国家高技术研究发展计划(2009AA01220); "核高基"重大专项(2009ZX01036-001-001)。

作者简介: 张家豪(1991—) 男,硕士研究生,主研方向为计算机体系结构; 单 征(通信作者),副教授、博士、博士生导师; 岳 峰,讲师、博士; 傅立国,博士; 王 军,硕士研究生。

收稿日期: 2018-01-17 修回日期: 2018-03-21 E-mail: 410541007@qq.com

库函数，从而提升二进制翻译系统的性能[2]。随着编程模块化和代码重用思想的普及，源代码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[3-5]。因此，通过对库函数调用的处理操作可以在保证原程序正确执行的基础上提高翻译代码的质量以及执行效率，对二进制翻译在国产平台下的应用具有重要意义。

本文针对国产申威平台处理器相关特性，设计源二进制文件中的库函数在国产目标平台下的解析处理机制。借助 helper 函数实现库函数替换，通过修改基本块划分规则，合并翻译基本块构建超级块，减少基本块数量与程序执行过程中基本块间跳转次数，以有效提高翻译效率。

1　相关工作

QEMU 是文献[6]编写的开源可变源和目标的高效跨平台的动态二进制翻译系统。QEMU 在用户模式下，支持多源平台和多目标平台。QEMU 采用指令集无关中间表示 TCG，可实现将 X86、ARM、Alpha、PowerPC 等平台的指令序列翻译为 TCG 中间表示，再翻译为目标指令集。TCG 技术屏蔽了不同平台之间的差异，是 QEMU 跨平台性的关键技术。QEMU 系统架构如图 1 所示。

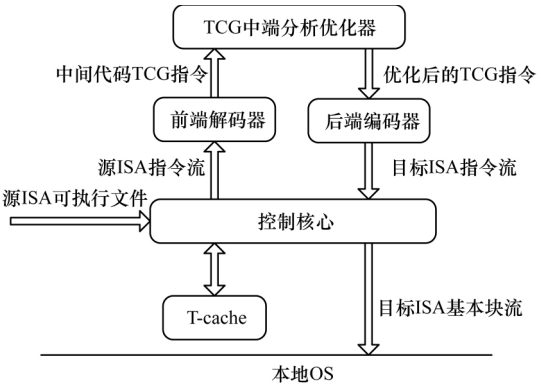

图 1　QEMU 系统架构

目前 QEMU 在对部分主流平台的翻译中已经达到了较好的执行效率。同时，由于 QEMU 代码开源性、翻译高效性以及可变源和目标特性等优势，使得其成为热门的研究动态二进制翻译工具之一。

申威处理器或申威 CPU，简称“SW 处理器”，SW 处理器源自于 DEC 的 Alpha 21164，采用基于 RISC 的自主指令集，具有完全的自主产权，其研制得到了国家“核高基”专项资金支持[7]。在国家“核高基”重大专项支持下采用自主指令集，是具有完全自主知识产权的处理器系列。本课题组在 QEMU 基础上，结合 SW 处理器平台相关特性，设计了针对 SW 平台的 TCG 后端代码生成翻译模块，同时构建设计了 Static QEMU，改造 QEMU 的解码模块，按照基本块划分的信息进行解码，生成基本块的 TCG 中间表示，使 QEMU 契合 SW 处理器平台，实现基本的二进制翻译功能。

2　QEMU 库函数处理过程

类似于处理器的执行过程，动态二进制翻译过程可以概括为查找、翻译、执行 3 个子过程[8-9]。而库函数处理区别于常规的动态二进制翻译过程主要体现在查找和翻译 2 个过程。

图 2 中的实线部分展示了库函数在常规动态二进制翻译中处理过程。在动态二进制翻译中，按照一般的翻译处理方式，会从 call 指令开始，参照指令的操作语义将指令用中间语言表示，再使用后端将其转换成可执行代码。这种方式使得源程序与翻译后程序保持状态一致，可以有效保证翻译程序的正确性。

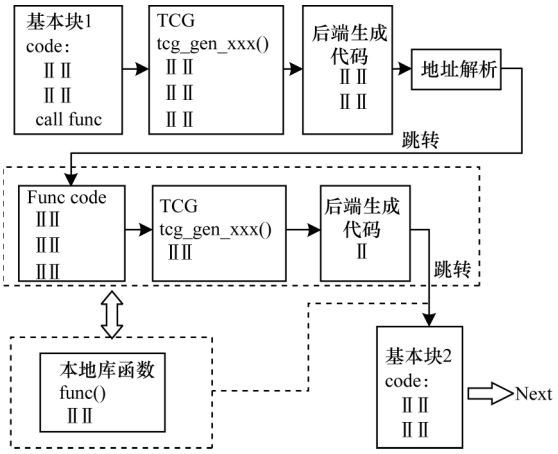

图 2　库函数替换与一般动态二进制翻译流程

图 2 中的虚线部分展示了使用库函数本地化处理的翻译方式，在翻译中会将 call 指令翻译成特殊的中间表示，这种中间表示不使用平台相关的后端进行代码生成，而是调用翻译器提供的模板函数作为翻译结果去执行。在使用库函数处理的方式翻译 call 指令时，先解析该地址的语义，查询对应的函数名，然后查找相应的处理函数，最终完成共享库函数调用全过程的翻译。

使用本地编译优化的模板处理函数，其代码的执行效率远远超过一般指令序列动态翻译所生成代码的执行效率，同时还节约了相应指令序列的翻译时间。相较于传统动态二进制翻译模式，库函数替换具有代码膨胀率低、翻译效率高的优势。

QEMU 使用基本块作为翻译的基本单元，划分基本块一般以跳转指令作为基本块的结束部分。在程序翻译中采用边执行边翻译的模式，在执行完一个基本块后，需要判断下一个基本块是否已经被翻译[10-11]。如果基本块已经被翻译，那么就直接执行 T-Cache 中翻译后的基本块，否则必须翻译基本块，然后才能执行[12]。无论是传统二进制翻译还是库函数替换翻译流程，在一次库函数翻译中，至少需要翻译 3 个基本块，程序每执行一个基本块就需要从执行状态切换出来进行判断，然后再进入执

行或翻译状态，这种频繁的跳转对翻译的效率有一定影响。

在当前关于库函数本地化的研究中，主要是从函数的快速识别与替换方面入手[2]，但是库函数的查找和替换时间不可避免。因此，本文拟从库函数处理基本块间的跳转入手，通过基本块的合并减少基本块加载次数，优化程序翻译执行时间，提升翻译效率。

3 面向 SW 平台库函数的处理方案

3.1 helper 函数的替换翻译模式

QEMU 使用 helper 机制对库函数进行辅助翻译处理，其以 C 语言作为基本语言，获取解析后的库函数参数后进行本地库函数的替换操作并在翻译完毕后返回，保证上下文一致性，确保翻译的正确执行。其基本工作流程如图 3 所示。

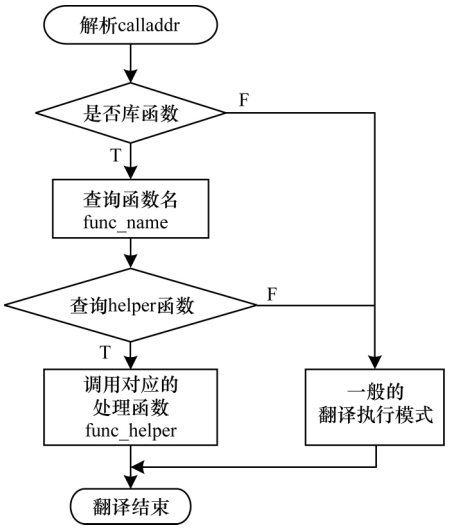

图 3 优化后翻译流程

在一次完整库函数处理的翻译过程中，需要进行 2 次查询判断:

1) 根据 call 指令信息进行解析，查询 addr 对应的函数名 func_name，并对该函数名进行判断，如果满足本地库函数替换条件，则转入判断 2)，反之则进入一般二进制翻译流程。

2) 将原函数的参数信息进行保存，查询名为 func_name 的库函数对应的处理函数 func_helper，如果查询到函数存在本地函数表中，则 helper 函数读取传递的参数信息并实现本地库函数对原函数的替换，之后对翻译环境进行恢复。如果未查询到相应函数，则进入一般二进制翻译流程。

helper 函数主要实现以下 3 个功能:

1) 环境模拟: 接收传参数，处理参数信息。

2) 目标替代: 调用库函数并形成相对应的后端二进制代码段，相当于生成了一个翻译完成的基本块后端代码，此功能是 helper 函数的核心功能。

3) 处理返回值: 按照目标机返回值的约定取出返回值，然后按照源系统约定放入到目标机模拟源系统的模拟寄存器或内存中[3]。

通过分析 helper 函数，可以在保留其核心功能的情况下对传参信息和返回值进行处理，构建基本块合并翻译框架。

3.2 基本块合并框架构建

如图 4 所示，左侧是传统二进制翻译和未修改的库函数基本块划分方式，每当遇到 call 指令就作为一个基本块结尾，一次库函数调用要将源代码段划分为 3 个基本块依次加载到内存中进行翻译操作，上一部分翻译完成后再进行之后的翻译。而如图 4 右侧所示的方法中，通过对 helper 函数的修改，如果函数解析后可以进行库函数替换操作，当翻译中遇到 call 指令时，对之后的内容进行分析处理，如果得出的函数为库函数，则不进行基本块划分，而是查询 helper 函数表找到库函数对应的 helper 函数替换原有地址。前后的基本块可合并为一个完成的“超级块”在翻译过程中只需要一次加载，同时节省了部分地址查找跳转的时间开销，提高了翻译效率。

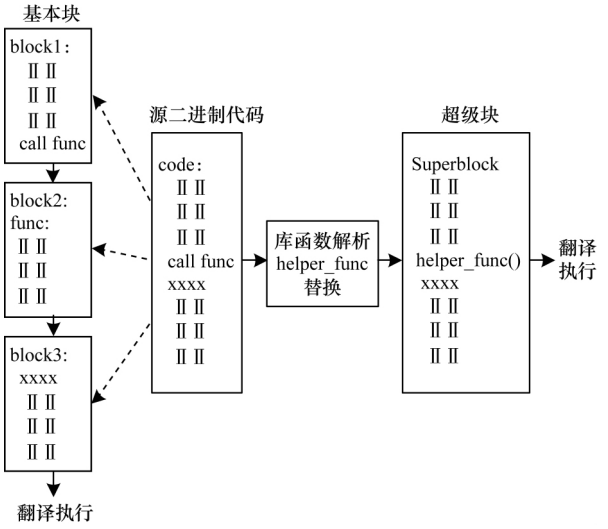

图 4 基本块合并翻译框架

3.3 适应平台的函数解析查找优化

3.3.1 函数名称解析

在源程序中使用 call 指令对库函数进行调用，以 ELF 文件格式为例，每一个 ELF 动态链接库和使用动态链接库的可执行程序都有一个过程链接表 PLT，PLT 表的每一项都是一小段代码，对应本运行模块要引用的一个全局函数。程序对某个函数的访问都被调整为对 PLT 入口的访问。每个 PLT 入口项对应一个 GOT 项，执行函数实际上就是跳转到相应 GOT 项存储的地址，该 GOT 项初始值为 PLT n 项中的 push 指令地址[13]。进行函数调用时先在 PLT 表中获取函数的入口地址后再从 PLT 表中跳到真正的函数入口，在确定 ABI 规范下，外部链接函数名称的解析过程是固定的，其共享库函数名称解析

过程如图 5 所示。

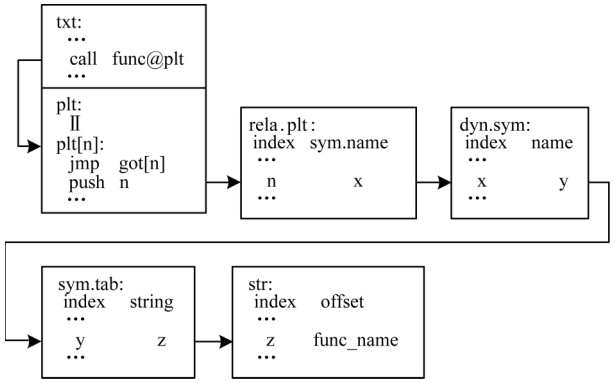

图 5　共享库函数名称解析过程

翻译程序首先根据 PLT 入口的序号 n 在 rela. plt 中查询符号名的索引，然后依次在 dyn. sym 段、sym. tab 段和 str 段中索引查询，最终得到外部链接函数名称的字符串，用以查询本地库替换。

3.3.2　函数查询优化

对于查询成功的库函数，在 2 次查询（plt_entry 查询、翻译器支持库函数处理的函数名称查询）以及 1 次解析（使用 plt_entry 解析函数名称）过程中，需要使用若干张查询表。特别是在共享库函数名称解析的过程中，需要基于文件规范从多个表格中进行多次索引查询获取函数名称。

基于以上原因，本文构建查询表 func. lib 与 func. info，其中，func. lib 是函数名称查询表，func. info 是函数具体信息表，两者之间是一一映射关系，只需要在 func. lib 中查询到函数名称就可对应到 func. info 中进行相应 helper 函数调用。其中 func. lib 使用哈希表结构，查询复杂度为 1，相较于遍历查找复杂度 n 与二分查找复杂度 lb n，可以有效减少函数查询时间，提升程序翻译执行效率。

4　算法设计与实现

修改后的 helper 函数工作流程如下：

步骤 1　获取函数名称找到指定 helper 函数后，可以通过源代码获取到参数信息，之后加载到 helper 函数指定位置作为参数信息。

步骤 2　将 helper 函数生成的后端代码替换原有 call 指令，并删除原有的传参信息。

步骤 3　修改返回值信息，使程序跳转到下一条待执行指令。经过修改程序的算法伪代码如下所示。

```
输入　call_addr [M],func_name [N]
1. main{
2. //主要处理过程
3.    for each call_addr do
4.        if call_addr [i]  can be found in func. lib
5.            if func_name [i]  can be found
6.                  //超级块构建，库函数替换
7.              call exchange_func( )
8.              call helper ( func_name)
9.            else
10.              //常规翻译模式
11.                 call tcg_trans( call_addr)
12.           end if
13.       end for
14.     }
15.     func helper( func_name) {
16.        call transfer_parameter ( )
17.        translate( )
18.        delete_parameter ( )
19.        return
20. }
```

其中，第 1 行 ~ 第 14 行为主要翻译流程，第 14 行 ~ 第 20 行为 helper 函数内具体功能。通过下面的例子进行详细分析，源系统为 X86 架构，源二进制代码和生成代码核心部分如图 6 所示。

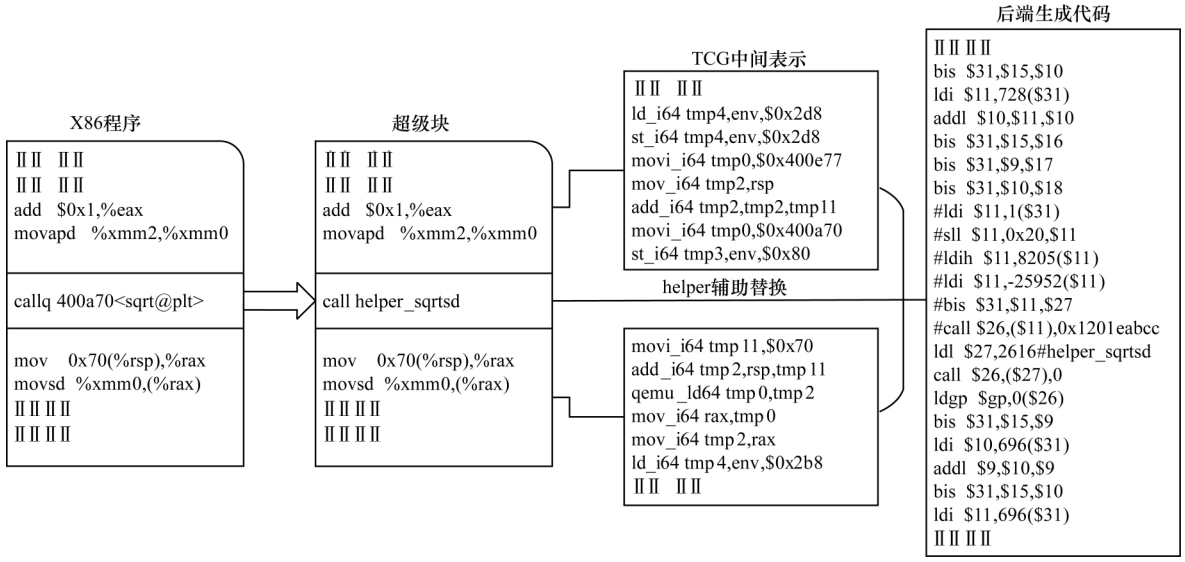

图 6　核心代码生成

在程序翻译中，遇到调用库函数指令“call q400a70”，通过解析查询找到与之相对应的翻译函数 helper_sqrtsd，对原有 call 指令进行替换操作实现超级块的构建。翻译超级块时，其余指令按照传统的 QEMU 翻译方式先转换为 TCG 中间代码再进行后端代码生成，call 指令调用 helper 函数直接生成后端代码。由于原传参信息已经传入 helper 函数中进行函数执行，因此要注释掉后端代码中相关代码。按照此翻译流程，该 x86 代码段只需一次翻译即可，减少了基本块间的跳转次数，同时减少了基本块间的参数保存于传递冗余指令操作，翻译产生的后端代码比原翻译模式少，程序的翻译执行效率提高。

5 实验结果与分析

为进一步验证优化的效果，在动态二进制翻译器 QEMU 上实现了库函数处理的翻译机制，在此基础上实现了本文提出的优化方法，最后在 SW 平台上进行了实验验证。关于库函数处理机制的正确性验证在课题组之前的研究中已经通过了相关的正确性测试[14-15]。

5.1 实验环境

实验测试所使用的环境如表 1 所示。

表 1 二进制翻译环境

配置	源平台	目标平台
OS	Fedora 2.6.27.5-117.fc10.i686	中标麒麟 4.0.0
CPU	Intel(R) Core(TM) 2 Quad CPU Q9500 @ 2.83 GHz	SW410
编译器	gcc-5.1.1	gcc-5.1.1

实验使用的翻译器为 QEMU-2.5.1，通过修改，支持标准 C 库中 math.h、stdio.h 和 string.h 下定义的大部分函数的库函数处理。

实验使用的测试集来自标准性能测试集 spec cpu2006。具体的测试用例包括 spec cpu2006 中所有的 C 程序，如表 2 所示。

表 2 spec cpu2006 测试用例

测试用例	任务
401.bzip2	Compression
403.gcc	Compiler
429.mcf	Combinatorial Optimization
445.gobmk	Artificial Intelligence: go
456.hmmer	Search Gene Sequence
458.sjeng	Artificial Intelligence: chess
462.libquantum	Physics: Quantum Computing
464.h264ref	Video Compression
433.milc	Physics: Quantum Chromodynamics
470.lbm	Fluid Dynamics
482.sphinx3	Speech recognition

5.2 结果分析

库函数本地化是基于动态二进制翻译模式进行的优化，与静态二进制翻译生成翻译后的可执行文件不同，动态二进制翻译将程序基本块作为基本单位，采用边翻译边执行的模式，因此，可以将程序翻译执行时间作为衡量翻译效率的指标。

实验对每个测试用例翻译执行的时间进行 3 组测试，每组测试的结果取 5 次执行时间的算数平均值。第 1 组测试使用一般的动态二进制翻译方式；第 2 组测试使用未优化的库函数处理翻译方式；第 3 组测试使用基本块合并优化库函数处理翻译方式。

为更直观地反映翻译效率变化，本文引入加速比 S_p，其计算公式如下：

$$S_p = T_{native} / T_{translated}$$

其中，T_{native} 为基本时间，$T_{translated}$ 为优化后的时间。本文实验将第 1 组作为基本时间，图 7 是测试环境下第 2 组、第 3 组相对第 1 组的加速比。

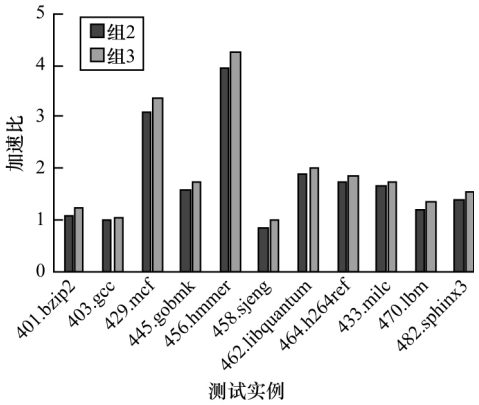

图 7 实验加速比对照

通过 spec 中 2 组加速比的对照可以发现，经过基本块合并后的库函数处理翻译方式，相较于未优化的库函数处理翻译方式，平均加速比有 9% 左右的提升。特别是对于部分有负加速的测试用例，该方式也具有较好的加速比。

6 结束语

本文对国产 SW 平台二进制翻译中库函数处理翻译模式进行优化，设计一种源二进制文件中库函数在国产目标平台下的解析处理机制。结合 QEMU 中的 helper 机制构建超级块，对 helper 函数进行修改，使其能够对特定的本地库函数进行解析，并根据国产 SW 平台的特性，实现由源程序向特定平台的转换。通过基于动态二进制转换器 QEMU 的实验验证，结果表明，该方法针对库函数查询开销的优化是有效的。

（下转第 83 页）

4　结束语

本文提出一种基于 Actor 模型的软总线构建方法，该软总线实时性强、扩展性高、能够同时等待多路消息、支持同步及异步消息通信，针对消息并发处理场景，能够极大提高通信效率，为网络化嵌入式系统的仿真测试提供有力支持。下一步将优化软总线的适配功能，支持半实物仿真环境的接入，以实现集半实物、全数字为一体的仿真测试平台。

参考文献

[1] VOLGYESI P，LEDECZI A. Component-based development of networked embedded applications [C] //Proceedings of the 28th Euromicro Conference. Washington D. C.，USA：IEEE Press，2002：68-73.

[2] ZHANG Jindong，QIN Guihe，XUN Yang，et al. Communication analysis and synthesis of distributed embedded network system [C] //Proceedings of IEEE International Sympo-sium on Embedded Computing. Washington D. C.，USA：IEEE Press，2008：345-349.

[3] DAMLE N S，KESKAR A G. Co-verification of networked embedded system [C] //Proceedings of International Conference on Emerging Trends in Engineering and Technology. Washington D. C.，USA：IEEE Press，2008：1340-1344.

[4] 张毅，胡勤友，施朝健. HLA 与 MAS 在分布式应用领域的仿真比较 [J]. 计算机技术与发展，2006，16(1)：150-153.

[5] 张大海，赖兰剑，陈鼎才. DDS 在分布式系统仿真中的应用 [J]. 计算机技术与发展，2011，21(3)：250-253.

[6] 高宇翔. Actor 模型在工业流水线控制系统的应用 [D]. 广州：华南理工大学，2012.

[7] 陈昊，高楚舒，魏峻，等. 基于 Actor 模型的高性能发布式 XMPP 服务器 [J]. 计算机系统应用，2015，24(10)：62-67.

[8] DENG F，GAO Feng. P2P-based full digital co-simulation and verification system design [C] //Proceedings of IEEE International Conference on Software Quality. Washington D. C.，USA：IEEE Press，2016：222-227.

[9] 李侃，王兵山. Actor 模型的继承机制研究 [J]. 计算机工程与科学，1995，17(1)：12-14.

[10] 董哲，刘琳，田籁声. 基于 ACTOR 模型的并发面向对象语言 AC ++ [J]. 软件学报，1997，8(3)：197-203.

[11] 刘海峰，王佳琪，梁星亮. 基于多核多线程的 HECC 并行算法的实现与分析 [J]. 陕西科技大学学报，2019，37(2)：167-172.

[12] 丁筱春，丁菁. 基于软总线的核心调度的快速实时处理 [J]. 计算机应用研究，2000，20(6)：70-72.

[13] SUBRAMONIAN V，XING Guoliang，GILL C，et al. Middleware specialization for memory-constrained networked embedded systems [C] //Proceedings of the 10th IEEE Real-time and Embedded Technology and Applications Symposium. Washington D. C.，USA：IEEE Press，2004：306-313.

[14] 陆庆，周世杰，秦志光，等. 消息队列中间件系统中消息队列与消息分发技术研究 [J]. 计算机应用研究，2003，20(8)：51-53.

[15] 蒋一新，孙涌. Peer-to-Peer 消息中间件的研究与设计 [J]. 计算机工程与科学，2006，28(11)：96-99.

编辑　吴云芳

(上接第 76 页)

参考文献

[1] ALTMAN E R，KAELI D，SHEFFER Y. Welcome to the opportunities of binary translation [J]. Computer，2000，33(3)：40-45.

[2] 杨浩，唐锋，谢海斌，等. 二进制翻译中的库函数处理 [J]. 计算机研究与发展，2006，43(12)：2174-2179.

[3] 谢海斌，张兆庆，武成岗，等. 二进制翻译中系统库函数的分类处理方法 [J]. 计算机应用研究，2008，25(4)：1057-1064.

[4] LIANG Yi，SHAO Yuanhua，YANG Guo，et al. Register allocation for QEMU dynamic binary translation systems [J]. International Journal of Hybrid Information Technology，2015，8：25-39.

[5] HAWKINS B，DEMSKY B，BRUENING D，et al. Optimizing binary translation of dynamically generated code [C] //Proceedings of IEEE International Symposium on Code Generation and Optimization. Washington D. C.，USA：IEEE Press，2015：68-78.

[6] BELLARD F. A fast and portable dynamic translator [C] //Proceedings of Annual Conference on USENIX Annual Technical. Washington D. C.，USA：IEEE Press，2005：125-136.

[7] 芮雪，王亮亮，杨琴. 国产处理器研究与发展现状综述 [J]. 现代计算机(普及版)，2014(8)：15-19.

[8] MICHEL L，PETROT F. Dynamic binary translation of VLIW codes on scalar architectures [J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems，2017(99)：1.

[9] WU Xia，CHENG Wen. Fast optimization of binary clusters using a novel dynamic lattice searching method [J]. Journal of Chemical Physics，2014，141(12)：124-130.

[10] D'ANTRAS A，GORGOVAN C，GARISDE J. Optimizing indirect branches in dynamic binary translators [J]. ACM Transactions on Architecture and Code Optimization，2016，13(1)：1-25.

[11] YOHN J W，BAUMAN M A，KAO F J，et al. Operand and limits optimization for binary translation system：US9021454 [P]. 2015.

[12] 吴浩. 二进制翻译系统 QEMU 的优化技术 [D]. 上海：上海交通大学，2007.

[13] 宁涛，王铮，龙川. 面向一般应用的动态加载机制 [J]. 计算机工程，2008，34(17)：79-81.

[14] 谭捷，庞建民，单征，等. 二进制翻译中冗余指令优化算法 [J]. 计算机研究与发展，2017，54(9)：1931-1944.

[15] 戴涛，单征，卢帅兵，等. 基于优先级动态二进制翻译寄存器分配算法 [J]. 浙江大学学报(工学版)，2016，50(7)：1338-1346.

编辑　索书志