软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.007099]

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

二进制翻译技术研究综述*

谢汶兵[1], 田雪[1], 漆锋滨[2], 武成岗[3], 王俊[1], 罗巧玲[1]

[1](无锡先进技术研究院,江苏无锡 214083)

[2](国家并行计算机工程技术研究中心,北京 100080)

[3](中国科学院 计算技术研究所,北京 100190)

通讯作者: 漆锋滨 qifb116@sina.com

摘 要: 随着信息技术的快速发展,涌现出各种新型处理器体系结构.新的体系结构出现为处理器多样化发展带来机遇的同时也提出了巨大挑战,需要兼容运行已有软件,确保较为丰富的软件生态群.但要在短期内从源码编译构建大量生态软件并非易事,二进制翻译作为一种直接从二进制层面迁移可执行代码技术,支持跨平台软件兼容运行,既扩大了软件生态群,又有效降低了应用程序与硬件之间的耦合度.近年来,二进制翻译技术研究取得了较大进展.为总结现有成果并分析存在的不足,本文首先介绍了二进制翻译技术的分类以及典型的二进制翻译系统,之后从指令翻译方法、关键问题研究、优化技术等方面分别进行分析总结,接着阐述了二进制翻译技术的核心应用领域,最后对二进制翻译技术的潜在研究方向进行展望.

关键词: 二进制翻译;翻译效率;等价变换;软件迁移;多融合优化

中图法分类号: TP311

中文引用格式: 谢汶兵,田雪,漆锋滨,武成岗,王俊,罗巧玲. 二进制翻译技术研究综述.软件学报. http://www.jos.org.cn/1000-9825/7099.htm

英文引用格式: Xie WB, Tian X, Qi FB, Wu CG, Wang J, Luo QL. Overview on Binary Translation Technology Research. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/ 1000-9825/7099.htm

Overview on Binary Translation Technology Research

XIE Wen-Bing[1], TIAN Xue[1], QI Feng-Bin[2], WU Cheng-Gang[2], WANG Jun[1], LUO Qiao-Ling[1]

[1](Wuxi Institute of Advanced Technology, Wuxi, Jiangsu 214083, China)

[2]((National Research Center of Parallel Computer Engineering and Technology, Beijing 100190, China)

[3](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

Abstract: With the rapid development of information technology, a variety of new architectures have emerged. The emergence of new architectures brings opportunities for the diversification of processors, and at the same time poses great challenges, which requires compatible operation of existing software to ensure a rich software ecosystem. However, it is impractical to manually compile large amounts of source code in a short time. As a technology that migrates executable code directly from the binary level, binary translation supports cross-platform software compatible operation, which not only expands the software ecosystem but also effectively reduces the coupling between applications and hardware. In recent years, the research on binary translation has made great progress. In order to summarize the existing achievements and the shortcomings, this paper first introduces the classification of binary translation technology and typical binary translation systems, then analyzes and summarizes the instruction translation methods, key issues and optimization techniques, then expounds on the core application fields of binary translation technology. Finally, we look forward to the potential research directions of binary translation technology.

Key words: binary translation; performance efficiency; equivalency transformation; software migration; multi-fusion optimization

收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05

随着信息技术的快速发展,对硬件算力要求不断提高,推动着各种新型处理器架构的涌现.然而一款处理器的推广,离不开丰富的软硬件生态支持.考虑到软件开发周期和开发成本,新型处理器很难在短时间内构建出毗邻 x86 和 ARM 架构的丰富软件生态群,存在市场竞争力弱、推广难度大的难题;另一方面,得不到广泛应用和市场份额的处理器也很难得到广大开源社区对其生态的维护支持.软硬件生态的滞后已成为制约新型处理器推广的关键瓶颈.在这种情况下,将 Wintel(Windows+Intel)和 AA(ARM+Android)等已经成熟的体系结构应用软件迁移到新型处理器架构,既加速了其生态构建,又促进了处理器研发的更新迭代.二进制翻译技术作为这样一种软件迁移手段,实现不同架构之间的应用软件兼容运行,其在提高硬件平台可用性的同时降低了软件开发和维护成本.纵观计算机行业的发展历程,利用已有平台的软件丰富新型处理器生态是业界常用的手段.自 20 世纪 90 年代开始,多种宿主机指令系统利用 x86 和 ARM 等平台丰富的应用软件生态群,推出了大量商用和开源的二进制翻译系统.例如 Digital FX!32[1]用于 Alpha 宿主机系统、Aries[2]用于 HP 的 PA-RISC、IA-32 EL[3]用于 Itanium 宿主机、Transmeta CMS[4]用于 Transmeta VLIW 指令系统、Rosetta 2[5]用于苹果 Apple Silicon M1、DAISY[6]和 BOA[7]用于 IBM 的 Power 体系结构.

除了软件迁移外,在虚拟化技术广泛应用的今天,二进制翻译也是跨架构虚拟机实现的核心技术,有效屏蔽了硬件差异.此外,移动计算和云计算催生出多样化异构系统,计算平台由传统的单一指令系统向多指令系统融合发展.例如 Qualcomm 和 ARM 将异构计算应用于移动设备,Google 使用多种指令系统组成大型机群.二进制翻译技术可以实现多指令系统的处理器融合,其在多样化异构系统中也得到了广泛应用.二进制翻译技术也被广泛应用在程序分析与优化、二进制符号执行、安全研究等领域.

综合来看,经过多年发展,二进制翻译技术已取得了良好进展,对多个领域的研究起到了促进作用.然而,二进制翻译技术依旧面临着翻译效率不高、目标体系结构硬件特性发挥不足、被翻译程序完备性保证难度大等挑战.如何探索更优的翻译方法、追求更高的翻译效率和覆盖更丰富的应用领域是业界持续关注的热点.

本文在 Web of Science Core Collection(WOSCC)和中国知网(CNKI)中分别使用关键词“binary translation”“虚拟化(+)二进制翻译”“binary translation*(QEMU(+)virtualization(+)LLVM)”“ISA*BT”搜索,并且人工筛选与“二进制翻译”“binary translation”密切相关的工作,统计文献检索情况如图 1 所示.可以看出,近十年文献发表数量明显增多,原因可能是:(1)硬件资源极大丰富与 CPU 性能过剩为二进制翻译提供了良好的硬件支撑.(2)虚拟化、QEMU、LLVM、编译优化等技术的发展为二进制翻译提供了技术研究的新机遇.(3)处理器多样化快速发展以及各种新兴领域对二进制翻译的需求不断增加.

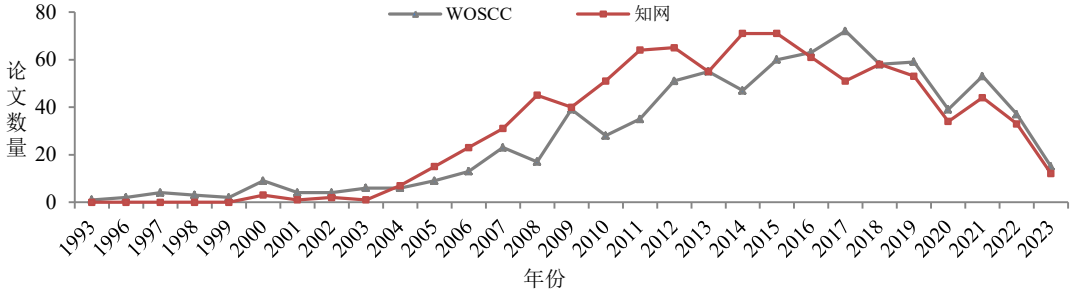


图 1 WOSCC 与知网中检索二进制翻译技术相关文献数量

本文关注到,在已经发表的学术研究中,针对二进制翻译的综述类文章并不多见.Li 等人[8]于 2007 年对动态二进制翻译与优化技术进行了概括总结,但其并未对多线程翻译、应用领域、指令翻译方法等方面进行分析,同时由于文献发表时间较早,大量最新技术并未涉及.Spink 等人[9]于 2020 年总结梳理了常见的动态二进制翻译系统,但其目的是与 Captive 系统进行对比.本文整理了 196 篇与二进制翻译主题密切相关的文献,时间跨度从 1994 年至 2023 年.通过深入分析总结二进制翻译相关技术和典型翻译系统,旨在揭示二进制翻译技术的研究进展、研究热点以及未来的研究方向.

本文从二进制翻译过程中最为关注的指令翻译、关键问题、性能优化和应用领域四个方面开展了综述.

全文组织结构如下:第 1 节简述了二进制翻译技术基本原理,然后从不同角度对二进制翻译技术做了分类,并梳理了二进制翻译发展历史上具有重要突破的典型系统;第 2 节总结了二进制指令翻译方法,将翻译方法划分为定制化中间表示的指令翻译、编译器框架辅助的指令翻译、规则匹配的指令映射翻译;第 3 节总结了二进制翻译技术面临的一些关键问题,包括存储单元映射、原子操作时序维护、异常和中断处理、内存一致性保证等;第 4 节讨论了二进制翻译优化技术,重点从翻译开销优化、运行时优化和代码生成优化总结二进制翻译常用的优化方法;第 5 节归纳了二进制翻译技术的典型应用领域,例如软件迁移、程序分析与优化、二进制符号执行等;第 6 节对二进制翻译技术的潜在研究方向进行展望;第 7 节对全文工作进行总结.

1 二进制翻译技术与典型系统

1.1 二进制翻译技术基本原理

严格来说,二进制翻译技术有狭义和广义之分.狭义的二进制翻译技术是指二进制翻译器本身,主要功能是实现源平台(程序被翻译之前的运行平台,也叫源机器平台或客户平台)程序指令到目标平台(程序被翻译之后的运行平台,也叫目标机器平台或宿主平台)指令的等价翻译;广义的二进制翻译技术则是一个复杂的翻译系统,其包含了指令翻译、运行环境模拟、目标代码执行、性能优化、异常处理等环节.广义的二进制翻译全面地描述了程序模拟过程.本文讨论的二进制翻译特指广义的二进制翻译技术.

图 2 给出了典型的二进制翻译系统基本框架图,核心模块包括程序加载、二进制翻译器、控制器、运行环境模拟、目标代码执行等.程序加载负责启动整个翻译系统,并将源程序代码加载到内存.二进制翻译器负责指令的翻译、优化、代码缓存等工作.控制器负责翻译和执行之间的控制流切换、缓存代码的更新与查找等.运行环境模拟负责系统调用、异常处理、存储单元镜像等模拟工作.目标代码执行则是在目标机器上执行翻译后代码.总结看来,二进制翻译技术的执行流程如下:首先,在目标机器平台加载源机器平台的二进制程序,并启动二进制翻译器.其次,二进制翻译器根据源程序代码逻辑进行解码、翻译、优化以及编码等指令翻译工作.最后,在目标平台执行翻译生成的二进制代码.

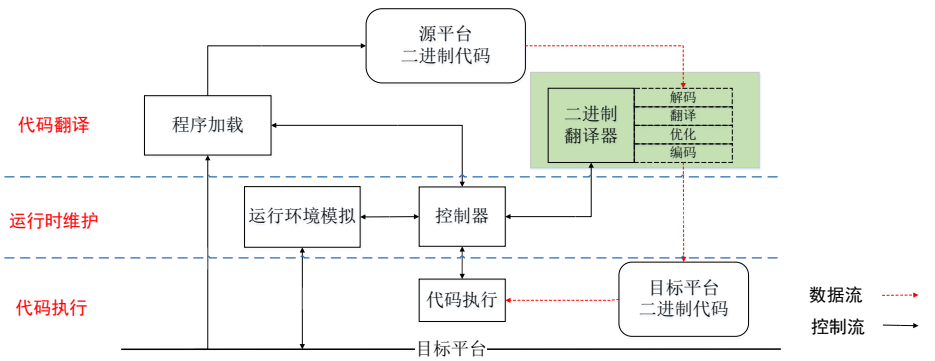


图 2 二进制翻译系统基本框架图(“彩印”)

1.2 二进制翻译技术分类

二进制翻译技术根据其输入的翻译对象、运行的翻译平台、选取的翻译方法等不同,我们从不同角度对其进行分类.

(1)根据翻译对象不同分类,分为系统级翻译和用户级翻译.系统级翻译的翻译对象是被翻译程序所在的整个系统,相当于在目标平台模拟一套与源平台完全相同的运行环境.翻译过程涉及对系统中所有进程、内存单元、外设、系统调用等全部环境的模拟,通常翻译效率较低.事实上,对于大多数应用来说,二进制翻译过程只需聚焦于应用程序本身,其运行的环境可以直接依赖于宿主机操作系统,由此便衍生出了用户级翻译.用户级翻译流程简单,通常翻译效率较高.

(2)根据翻译平台不同分类,分为同平台翻译和跨平台翻译.同平台翻译中源平台和目标平台的体系架构相同,二者在指令集、存储单元、内存模型等方面差异较小,模拟实现相对简单.同平台翻译更多聚焦于代码插桩与分析、架构向后兼容、新架构特性优化等研究;跨平台翻译中源平台和目标平台的体系架构不同,相比同平台翻译,跨平台翻译难度显著增加,翻译过程也更为复杂,是当前二进制翻译的研究热点.

(3)根据翻译时机不同分类,分为解释执行、静态翻译、动态翻译、动静结合.解释执行作为最简单的翻译方法,以单条指令为单位将源程序指令逐条实时解释执行,无需控制器频繁做控制流的切换.解释执行不保存已解释的指令,也不开展代码优化,执行效率通常要比本地原生执行慢 10 到 100 倍[10].当前先进的翻译系统中,完全基于解释执行设计的二进制翻译系统较为少见;静态翻译将指令翻译与代码执行过程分离,以输入的源程序为单位,离线完成指令翻译并充分优化生成的代码,执行效率较高.然而,静态翻译无法提前获知程序完备控制流信息,面临着一些技术挑战,例如翻译代码运行时对代码挖掘、代码重定位等问题处理不足,这也使得静态翻译在使用场景上受限,其在嵌入式应用中使用较多,并且通常与解释执行联合使用[2, 7, 11];动态翻译采取边翻译边执行的即时编译策略,以基本块或者函数体为单位开展翻译和执行,当遇到未翻译代码时控制流再切换至翻译器进行翻译.动态翻译弥补了静态翻译无法获取完备控制流信息的不足.同时,动态翻译还引入了代码优化和代码缓存,相比解释执行在执行效率有所提升.但是动态翻译面临着翻译开销大、代码优化不充分、控制流切换频繁等挑战.动态翻译是当前主流的二进制翻译方法和研究热点[5, 12-15];动静结合将动态翻译和静态翻译充分结合,基于静态翻译尽可能地预先离线翻译代码并缓存,然后在执行预先翻译代码时,如果遇到间接跳转等不确定情况再采用动态翻译进行补充翻译,动静结合内容将在 4.1 节进一步介绍.

1.3 典型二进制翻译系统

FX!32[1]是 Digital 公司开发的动静结合的二进制翻译系统,实现了 Win32/x86 应用在 Windows NT/Alpha 平台上的高效运行.FX!32 基于“剖析-优化”的翻译策略,首次将环境模拟、运行时信息生成和二进制翻译结合到一起.

BOA[7]是 IBM 公司开发的动态二进制翻译系统.为了兼容已有的 PowerPC 体系结构,BOA 将 PowerPC 指令转换为简单的 VLIW 操作原语,通过解释执行探测热路径代码并获得运行时信息来指导动态优化.

Aries[2]是 HP 公司开发的动态二进制翻译系统.Aries 使用解释器模拟执行源程序来发掘热路径代码,然后利用动态二进制翻译器翻译并优化热代码,最后缓存翻译生成代码.Aries 实现了 PA-RSIC 与 IA-64 之间的浮点寄存器映射,避免了大量潜在的寄存器存取操作.

Valgrind[16]作为同平台运行的二进制翻译与二进制插桩工具,采用“反汇编-再合成”的方式将二进制变换为 VEX IR 中间代码.Valgrind 基于影子内存映射的代码插桩与分析方法,在内存泄漏、程序分析、性能优化等方面发挥着重要作用.

QEMU[13]作为当前最为流行的多平台开源翻译框架,具有良好平台可扩展性,同时支持系统级和用户级翻译.QEMU 为了兼容多目标机和多宿主机,并未充分利用宿主机体系结构特征.未经优化的 QEMU 的翻译效率通常只有本地原生编译执行的 10%左右[17].近年来,针对 QEMU 涌现出大量研究,例如寄存器分配优化[18]、中间代码优化[18]、虚实地址转换优化[19]、基本块链接优化[20]、缓存管理[21]、多线程优化[20, 22]、分支跳转优化[23]、向量优化[24, 25]、Helper 函数优化[20, 26, 27]等.同时衍生出大量工具,例如基于 LLVM 优化加速的多线程翻译器 HQEMU[20]、缓存独占的并行全系统模拟器 COREMU[28]、缓存共享的并行全系统模拟器 PQEMU[29]、符号执行与二进制翻译融合的 SymQEMU[30]、基于分布式框架的 DQEMU[31]、兼容 Pin[32]的二进制插桩工具 PEMU[33]、面向程序并行优化的 LLPEMU[34]、动态二进制分析工具 PANDA[35]等.

CrossBit[36]是上海交通大学实现的多源多目标翻译系统,支持 MIPS、x86、Sparc 平台的应用到 Power、x86、Sparc 平台上翻译运行.CrossBit 采用 Vlnst 低层次虚拟精简指令集作为中间表示层以完成代码转换和优化.然而,CrossBit 对异常处理的支持存在不足.

Tango[37]实现了从 ARM32 应用到 ARM64 的翻译,由动态翻译器、预翻译器、执行环境三部分组成.Tango 充分利用了代码缓存和多线程机制,完整支持 ARM、VFP 和 NEON 指令集,并且支持数千个 Android 应用程序

的翻译运行.

ExaGear[12]是华为开发的动态二进制翻译系统,实现了 x86、x86-64、ARM 程序在 ARM64 平台的翻译运行.ExaGear 通过构建快速翻译+热点代码深度优化策略.采用 Trace 优化减少间接跳转查找开销,有效的改善了内存布局和数据局部性,充分发挥了鲲鹏的多核优势.

Instrew[14]是一个客户端+服务端的多进程二进制翻译和二进制插桩框架.相比于其他类似框架,Instrew 具有以下优势:(1)基于函数级粒度的翻译并且支持 SIMD(single instruction multiple data),对浮点指令的翻译效率较高.(2)被提权的中间表示与 LLVM IR 兼容,可以复用 LLVM 代码框架.(3)多进程的设计方法将翻译和执行部署在不同进程.然而,Instrew 对 x86 指令集、浮点异常以及舍入模式支持不足.

Rosetta 2[5]是苹果公司开发的商用二进制翻译系统,实现了 x86-64 应用在 ARM64 平台上的翻译运行.Rosetta 2 采用动静结合的代码翻译方法,同时在 ARM64 硬件上支持英特尔架构的内存模型,提升了对并发程序的翻译效率.

FEX-emu[38]是一套从 x86 和 x86-64 到 AArch64 的开源二进制翻译框架,致力于实现 Steam 和 Linux/x86-64 游戏在 AArch64 平台上高效运行.FEX-emu 基于 JSON 格式的 IR(intermediate representation)中间表示,支持将库函数调用转发到主机 OpenGL 驱动程序和其他组件中.

Houdini[39]实现了 Android/ARM 应用在英特尔处理器上的高效翻译运行.Houdini 使用复杂的动态指令生成机制进行性能优化,这使得翻译生成的指令极难精确与原生 ARM 指令映射.

LAT[17]是由龙芯中科开发的一套支持多平台翻译的动态二进制翻译系统,实现了 x86、ARM、MIPS 应用在 LoongArch 上高效翻译运行.LAT 通过软硬协同设计的方法提升二进制翻译效率,为上层应用软件提供了良好的虚拟运行环境.

Lasagne[40]静态二进制翻译器实现了一套支持并发推理的强弱内存一致性变换模型.Lasagne 基于 LLVM 框架设计实现,为不同内存模型处理器之间的并发二进制程序翻译与形式化验证提供了重要参考价值.

除了上述翻译系统,还存在着大量其他著名的二进制翻译系统,例如中科院计算所的 DigitalBridge[41]、面向全系统翻译的 Captive[9]以及基于“复制-再插桩”的 DynamicRIO[42]同源二进制插桩与分析工具等.本文梳理了二进制发展历史上具有重要技术突破的典型翻译系统,同时我们重点关注最近 10 年的研究工作,最终总结出如表 1 所示的典型二进制翻译系统列表.

表 1 典型二进制翻译系统

	文献	年份	源平台	目标平台	类型[1]	技术特点
用户级翻译	Shade[43]	1994	SPARC, MIPS	SPARC	D	指令缓存和 Trace 优化
	DAISY[6]	1997	RS/6000	VLIW	D	实现指令并行化.解决了内存一致性问题
	FX!32[44]	1998	IA-32	Alpha	H	首次将环境模拟、剖析数据生成和二进制翻译结合
	BOA[7]	1999	PowerPC	VLIW	D	硬件支持的精确异常处理.乱序执行提升指令并行度
	Aries[2]	2000	PA-RSIC	IA-64	D	支持浮点寄存器映射.对系统调用直接映射
	Strata[22]	2001	SPARC/MIPS/ IA-32	SPARC/MIPS	D	多线程翻译.提供丰富接口函数支持功能扩展
	Vulcan[45]	2001	IA-32, IA-64, MSIL	IA-32, IA-64, MSIL	H	支持静态和动态代码修改.支持跨组件分析与优化
	Bintrans[46]	2002	PowerPC	Alpha	D	不使用中间表示,人工分析源平台可执行文件
	Walkabout[47]	2002	SPARC v8	SPARC v8	H	JVM,解释执行
	Valgrind[16]	2003	多架构支持	多架构支持	D	基于 VEX IR 中间表达式转换.重量级代码插桩与分析工具
	DigitalBridge[41]	2004	x86	MIPS	H	内建解释执行模块.间接转移和标志位处理较优
	Dynamorio[42]	2004	PA-RISC, IA-32	PA-RISC, IA-32	S	代码缓存、块链接、Trace 构建等优化

[1] S:静态翻译, D:动态翻译, H:动静结合翻译

续表 1 典型二进制翻译系统

	文献	年份	源平台	目标平台	类型	技术特点
用户级翻译	QEMU[13]	2005	多架构支持	多架构支持	D	注册加载器钩子执行不同架构.
	IA-32 EL[3]	2006	IA-32	IA-64	D	支持 SIMD 指令,支持热路径优化
	StarDBT[48]	2007	IA-32, x86-64	IA-32, x86-64	D	用户级和系统级,支持 Trace 链延长优化
	CrossBit[36]	2008	Mips, x86, Sparc	Power, x86, Sparc	D	多线程翻译.基于 Vlnst 低层次的虚拟中间表示层
	EHS[49]	2009	ARC700	IA-32	D	多个基本块组成热代码优化
	Strata-ARM[50]	2009	ARM	ARM, IA-32	D	动态翻译应用于嵌入式系统.间接跳转优化
	ISAMAP[51]	2010	PowerPC	IA-32	D	灵活的跨指令集间指令直接映射
	ARCSim[52]	2011	ARC700	x86-64	D	基于并发任务的 JIT 编译执行热点代码
	Harmonia[53]	2011	ARM	IA-32	D	硬件指令加速状态标志位翻译
	HQEMU[20]	2012	x86-64, ARM64	x86-64, ARM64	D	多线程翻译与超级代码块优化
	LLBT[54]	2012	ARM	x86/x86-64, MIPS	S	基于 LLVM IR 中间转换.研究静态翻译中代码自修改、重定位问题
	SINOF[55]	2012	IA-32	IA-32	H	动静结合方式,有效减少信息收集和程序优化开销
	HERMES[56]	2015	IA-32	ARM, MIPS	D	提供后优化技术,利用目标平台架构特征挖掘指令之间数据依赖并开展冗余代码优化
	MAMBO-X64[57]	2017	AArch32	AArch64	D	硬件协同返回地址预测,锁指令翻译优化
	Rv8[58]	2017	RISC-V	x86	D	微指令合并.JIT 编译
	Tango[37]	2019	ARM	ARM64	H	预先翻译,支持 VFP 和 NEON 指令翻译
	DAOW[59]	2019	Android 平台	Windows PC 平台	D	跨操作系统转换,引入图形渲染优化
	LAT[17]	2020	x86-64, MIPS, ARM	LoongArch	D	软硬协同设计,多融合优化
	ExaGear[12]	2020	x86, x86-64, ARM	ARM64	D	内存布局和局部性优化,翻译效率高
	Rosetta 2[5]	2020	x86-64	ARM64	H	软硬协同设计,AOT&JIT 编译优化
	LLPEMU[34]	2021	x86-64	ARM	H	动静结合,多面体优化,循环并行化
	SymQEMU[30]	2021	x86-64, AArch64	x86-64, AArch64	D	基于二进制翻译的符号执行,灵活性强
	Instrew[14]	2021	AArch64, RISC-V	x86-64	D	基于 LLVM 客户端+服务端多进程翻译模式
	FEX-Emu[38]	2021	x86, x86-64	AArch64	D	基于 JSON 格式的 IR 表示,库函数本地化
	Houdini[39]	2021	ARM	x86	D	完整的寄存器映射.微码级翻译
	Box86[60]	2021	x86, x86-64	ARM64,RISC-V	D	系统库本地化,重写链接与加载工作
	Lasagne[40]	2022	x86	ARM	S	LIMM 强弱内存模型变换.形式化验证
	Risotto[61]	2022	x86	ARM	D	基于 TCG IR 转换的强弱内存翻译模型
系统级翻译	Embra[62]	1996	MIPS	MIPS	D	首个使用动态翻译技术的 DBT.块链接优化
	Transmeta CMS[4]	2003	IA-32	Custom VLIW	D	完整软硬协同设计.重组翻译后代码,支持指令并行化
	QEMU[13]	2005	多架构支持	多架构支持	D	基于 TCG IR 转换.可移植性和可扩展性强
	MagiXen[63]	2007	IA-32	IA-64	D	与 XEN 集成
	PinOS[64]	2007	IA-32	IA-32	D	全程序细粒度动态插桩
	PQEMU[29]	2011	ARM	x86-64	D	缓存共享的并行全系统模拟器.低开销
	COREMU[28]	2011	x86-64, ARM	x86-64	D	CPU 独占缓存的并行全系统模拟
	LIntel[65]	2012	IA-32	Elbrus	D	引入独立线程运行动态优化
	Pico[15]	2017	x86-64, AArch64	x86-64, POWER8	D	多线程翻译.硬件内存事务支持的锁指令翻译
	HybridDBT[66]	2019	RISC-V	VLIW	H	硬件协同加速
	Captive[9]	2019	ARMv8	x86-64	H	离线优化与 JIT 即时编译结合.裸金属环境运行
	Banshee[67]	2021	RISC-V	AMD EPYC	S	基于 LLVM 框架.多核系统中可扩展性好

2 二进制翻译技术中的指令翻译

二进制翻译器(如图 2 中绿底色标注)的核心功能是指令翻译,其根据要匹配的粒度从二进制程序中提取待匹配的函数、基本块或者指令,一一对应地模拟客户机的所有指令功能,实现指令的解码、翻译、优化和编码等工作,最终转换为目标平台机器码.但不同的指令翻译方法最终生成的代码质量存在差异.本文梳理表 1 中典型翻译系统所采用的指令翻译方法,将其总结为定制化中间表示的指令翻译、编译器框架辅助的指令翻译和规则匹配的指令映射翻译三种.

2.1 定制化中间表示的指令翻译

为了摆脱指令翻译时对架构的依赖,降低翻译难度,有研究提出针对二进制翻译器自身设计一套架构无关的统一中间表示(intermediate representation,IR),其将翻译工作转换为源平台指令到 IR 中间表示的代码提权和 IR 到目标平台指令的重新编码生成.统一的 IR 表示既降低了架构依赖性和指令翻译复杂度,又提升了二进制翻译系统向多平台的可扩展性与兼容性.该方法为函数语义特征提取带来了极大的便利.图 3(a)为基于定制化中间表示的指令翻译流程图,典型的中间表示如 TCG IR、VEX IR.

TCG IR 作为 QEMU 采用的独立于架构的抽象中间表示,支持 140 余种不同的操作类型.TCG IR 使用微指令操作解释源平台指令,每条微指令近似于原子操作,保证了翻译时的正确性.图 3(b)为基于 TCG IR 实现 ARM64 到 x86-64 的指令翻译过程.为了尽可能快的完成翻译工作,QEMU 并未对 TCG IR 开展深入优化,导致其翻译效率较低.考虑到翻译复杂度和平台可扩展性,TCG IR 主要面向内存访问、整数运算、逻辑运算等基础指令表示,对于浮点和向量等功能复杂的指令则通过引入 C/C++语言设计的 Helper 函数模拟实现.针对 TCG IR 在浮点和向量指令转换方面的不足,文献[25]和[68]设计了向量 TCG IR 来表示 SIMD 指令,不再依赖 Helper 函数实现,丰富了 TCG IR 对不同指令类型的表示.

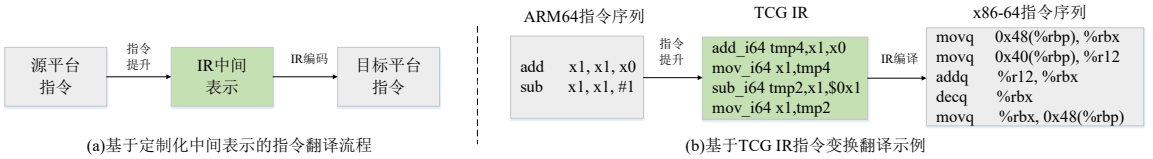


图 3 基于定制化中间表示的指令翻译

VEX IR 作为 Valgrind 采用的一种二地址形式的架构无关中间表示,支持 1000 余种不同的操作类型.VEX IR 基于 SSA (static single assignment)形式对每一个变量进行显式的类型变换.VEX IR 设计了表达式和语句申明来模拟存储单元读写、变量读写、条件赋值、特殊函数调用等操作.VEX IR 支持对多种平台的指令集进行统一描述,保证了不同平台的二进制代码直接复用插桩工具.VEX IR 目前对于 AVX、FMA 等向量指令支持不足.此外,为了保证多平台的兼容性,基于 VEX IR 表示的代码翻译效率较低.

除了上述定制化中间表示之外,还有其他基于 IR 的翻译设计,例如使用 REIL[69]作为中间表示的 IDA Pro[70],面向程序属性的形式推理 BAP[71],基于二进制语义学习的 LISC[72],面向 CrossBit 的低层次虚拟精简机器指令 VInst[36],支持 CPU/GPU 异构虚拟执行环境的中间表示 GVInst[73]等.

2.2 编译器框架辅助的指令翻译

定制化中间表示的指令翻译是针对二进制翻译器量身定做,具有很强的针对性.但是其也存在明显不足:需要开发人员手工开发编码、中间表示和解码过程,且在兼顾可扩展性的同时牺牲了翻译效率.考虑到二进制翻译的代码优化与常规编译优化具有相似性,大量编译技术可以直接被复用.为此,有研究提出利用编译器框架辅助完成二进制指令翻译,将编译器的中间表示和编译后端代码引入到二进制翻译中,从而充分利用编译器已有优化技术.此外,编译器在开源社区具有较高的活跃度,进一步促进了二进制翻译技术的更新迭代.图 4(a)为基于编译器框架辅助的指令翻译流程.

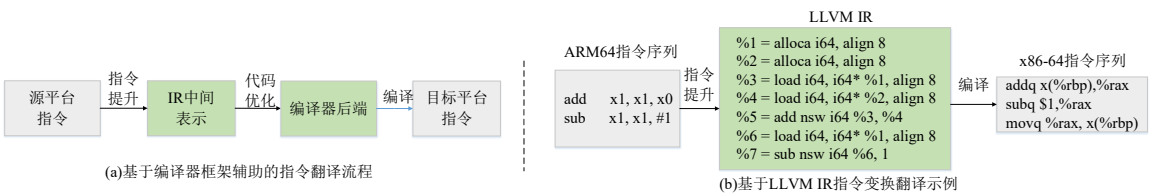


图 4 基于编译器框架辅助的指令翻译

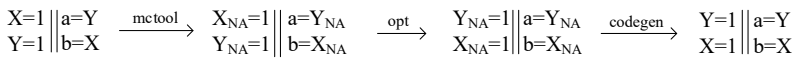
LLVM[74]作为一款模块化设计的优秀编译器,因具有丰富的优化机制而被广泛使用.近年来涌现出大批基于 LLVM IR 中间表示的二进制提升工具,表 2 对相关工具进行了总结.可以看出,从二进制代码到 LLVM IR 中间表示的转换工具较为丰富,这为二进制翻译技术与编译器框架的深度融合研究奠定了良好基础.

表 2 基于 LLVM IR 的提升工具对比总结 [2]

工具名称	社区维护	32 位架构	64 位架构	文件类型	浮点指令	向量指令
Remulle[75]	√	-	x86-64, AArch64, rv64	ELF	⭘	●
McSema[76]	√	x86, SPARC32	x86-64, AArch64, SPARC64	ELF, PE	⭘	⭘
Mctoll[77]	√	ARM, RISC-V32	x86-64, RISC-V64	ELF, PE, Mach-O	●	●
Retdec[3]	√	x86, ARM, MIPS, PIC32, PowerPC	x86-64, arm64	ELF, PE, Mach-O, COFF, AR, HEX, Raw	⭘	○
Revng[4]	×	x86, MIPS, ARM, s390x	x86-64, AArch64	ELF	○	○
Bin2llvm[5]	√	x86, ARM	-	ELF	○	○
Remill[6]	√	x86, SPARC32	x86-64, SPARC64, Aarch64	ELF, PE, Mach-O	⭘	●
Anvill[7]	√	x86	x86-64, Aarch64	ELF	○	○

近年来涌现出大量基于 LLVM IR 的二进制翻译研究,图 4(b)为基于 LLVM IR 实现 ARM64 加法与减法指令到 x86-64 平台的等价翻译示例.Hong 等人[20]提出将 LLVM IR 引入到 QEMU 中提出了 HQEMU,首先将二进制代码全部转换为 TCG IR,当探测到某一段代码块为热路径代码时,将相应的 TCG IR 转换为 LLVM IR 并进行深入分析与优化.然而,该方法受到 TCG IR 优化不足的限制,无法对未变换为 LLVM IR 的 TCG IR 开展细粒度优化.Chipounov 等人[78]提出将 QEMU 中间表示全部采用 LLVM IR 实现,之后将 LLVM IR 变换为 LLVM 字节码编译执行.由于该方法进行 LLVM IR 变换时开销较大,最终翻译效率低于 QEMU 原有的翻译方法.Shen 等人[54]提出的 LLBT 静态二进制翻译工具采用 LLVM IR 中间表示进行代码优化与指令生成,然后采用 LLVM 编译器将 LLVM IR 转换为最终目标平台代码,结果表明该方法比 QEMU 动态翻译性能平均提升了 7 倍.Rellume[75]首先将二进制代码提升到 LLVM IR 中间表示,之后再利用 LLVM 代码生成器完成编译,该中间表示充分兼容了 LLVM 编译器.Rocha 等人[40]提出的 Lasagne 翻译器基于改进版 mctoll[77]完成二进制逐层变换到 LLVM IR,先将二进制反汇编为低级中间表示 MCInst,再提升变换为 MachineInstr 并构建 CFG(control-flow graphs),最终变换为 LLVM IR 抽象代码.改进版 mctoll 支持浮点调用和尾部处理、SSE 指令以及奇偶校验等相关代码提升.You 等人[79]增强了 LLVM IR 对浮点指令的表示能力,扩展对非数处理、浮点异常和各种舍入模式的支持,使得 LLVM IR 对目标平台浮点指令特性的支持更加完善.

基于 LLVM IR 的中间表示充分利用了 LLVM 编译器的优化特性,在通用性和效率上相比于 TCG IR 具有显著提升.然而,也存在一些不足:(1)由于二进制文件缺少数据类型、控制流或函数调用抽象信息,导致其对浮点指令和向量指令提升支持不足.(2)对内存一致性模型的翻译支持不足.图 5 是基于 mctoll+LLVM 实现 x86-to-ARM(从 x86 到 ARM 架构的翻译)并发程序翻译的示例,X_{NA} 与 Y_{NA} 表示共享内存非原子变量的访问操作.图 5(a)中的代码在 x86 架构上执行时不允许出现 a=1,b=0 的输出结果,但是该代码经过 LLVM IR 中间变换后,最终在 ARM 架构上执行可能出现 a=1,b=0 的输出结果,主要原因是 LLVM IR 缺少对并发推理的支持.



(a)x86 (b)未优化LLVM IR (c)优化LLVM IR (d)ARM

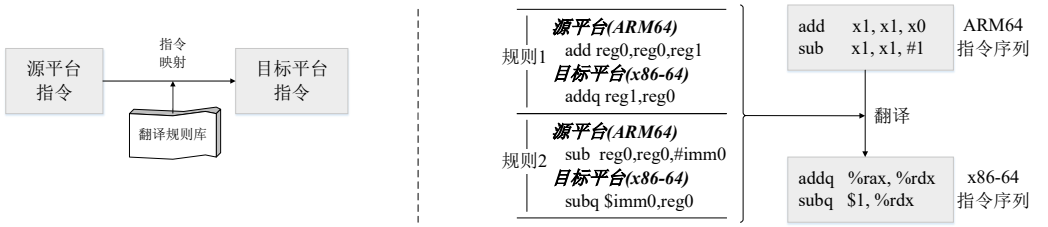
图 5 基于 LLVM IR 实现 x86-to-ARM 翻译

[2] ●支持, ⭘部分支持, ○不支持
[3] https://github.com/avast/retdec
[4] https://github.com/revng/revng
[5] https://github.com/cojocar/bin2llvm
[6] https://github.com/lifting-bits/remill
[7] https://github.com/lifting-bits/anvill

2.3 规则匹配的指令映射翻译

无论是定制化中间表示的指令翻译还是利用编译器框架辅助的指令翻译,二者均依赖于 IR 中间表示.这些方法在中间转换过程中损失了大量的程序语义,生成代码膨胀率高.例如,基于 VEX IR 的代码翻译后膨胀率达 10 倍以上[80].随着机器学习技术的不断发展,有研究将机器学习引入到指令翻译中,提出基于规则匹配的指令映射翻译方法,避免引入中间 IR 转换.将指令翻译变为规则训练、规则学习和规则映射的“训练-学习-生成”过程[81, 82].图 6(a)为基于规则匹配的指令映射方法的翻译流程,将源平台指令与预先形成的翻译规则库进行匹配,找到对应的目标平台指令完成指令映射.图 6(b)为基于规则匹配映射方法的示例,在该示例中通过匹配规则 1 和规则 2 实现 ARM64 加法与减法指令到 x86-64 平台的等价翻译.

在翻译规则库的形成上,Bansal 等人[83]首次将编译器超级窥孔优化技术[84]引入到二进制翻译的规则生成.与常规编译器优化不同,在学习阶段,Bansal 等人为源平台和目标平台指令序列构造一个等价翻译规则,之后将生成规则提供给静态二进制翻译器使用.测试结果表明,该方法使得 SPEC CINT2000 部分程序的执行效率超过本地原生运行.但是该方法采用近乎暴力搜索潜在匹配指令,指令步长较短,不支持对浮点指令的学习.针对基于规则映射方法对应用指令集覆盖率不足的问题,Wang 等人[81]混合使用基于规则映射与基于 IR 变换的翻译方法,优先采用规则映射方式完成指令翻译,对于无法映射翻译的情况再将指令变换为 TCG IR,该方法有效解决了规则匹配失效导致指令无法全覆盖的难题.Song 等人[82]提出在源码语义学习时引入动态滑动窗口尝试寻找更优的匹配规则,在翻译规则中允许客户机、宿主机之间存在 ISA(instruction set architecture)的语义差异,该方法相比文献[81]的指令覆盖率和翻译效率有显著提升.



(a)基于规则匹配的指令映射翻译流程 (b)基于规则匹配的指令映射翻译示例

图 6 基于规则匹配的指令映射翻译

Jiang 等人[85]分析了训练集规模与生成规则数量的关系,实验结果表明仅仅通过无限扩大训练集规模并不能保证指令集覆盖率持续提升,这也是基于规则匹配方法进行指令翻译面临的一个难点问题.Jiang 等人[85]使用参数化翻译规则学习方法进一步优化,根据指令集编码分类规律衍生出更多可用规则,对于被翻译的源平台指令,先参数化匹配规则再实例化.但参数化方法限制了其平台可移植性.Hasabnis 等人[72]反向利用 GCC 的指令模板描述来扩展学习规则,使用 GCC 编译大量源代码包,分析编译后端生成的 IL(intermediate language)片段与输出的汇编代码(assembly)之间的关系,形成映射规则<assembly,IL>,最终以 IL 为中间桥梁实现不同架构的指令映射,自动生成更多规则.实验结果表明其指令覆盖率达 99.5%,但是该方法仅支持编译器生成指令,无法处理手工加入的指令.

此外,也有研究提出将编译框架转换和规则映射相结合的代码生成方法.Xu 等人[86]提出 Copy-and-Patch 快速编译技术,实现从高级语言或低级字节码到二进制代码的高效转换.Copy-and-Patch 系统包含 MetaVar 编译器和 Copy-and-Patch 代码生成器两部分.MetaVar 预先构造满足多种场景的架构无关模板库.Copy-and-Patch 代码生成器根据用户输入的字节码或 AST(abstract syntax tree)的程序特征利用模式匹配选择最高效的模板库,最终生成高效的二进制代码.Copy-and-Patch 技术在代码生成效率和质量上优势显著,并且对不同的字节码和 AST 具有很强的可扩展性.二进制翻译作为具有特殊前端的编译器,有理由相信 Copy-and-Patch 技术对二进制指令高效翻译同样具有很强的借鉴意义.

2.4 指令翻译方法分析与对比

本节介绍了二进制指令翻译常用的三种翻译方法:定制化中间表示的指令翻译、编译器框架辅助的指令翻译和规则匹配的指令映射翻译.我们从代表性方法、翻译效率、平台可扩展性、代码膨胀率、不足等五个方面对三种指令翻译方法进行对比,结果如表 3 所示,可以看出三种方法各有千秋.然而,设计一种高质量的指令翻译方法是一项具有挑战性的工作,既要求在功能上完全表达被翻译指令,又要在翻译效率上足够高效.当前的二进制翻译系统设计,普遍做法是将不同的指令翻译方法融合使用[20, 81, 86].

表 3 二进制指令翻译方法对比

类别	代表性方法	翻译效率	平台可扩展性	代码膨胀率	不足
定制化中间表示的指令翻译	TCG IR[13]、VEX IR[16]	低	强	高	翻译工作量较大;优化手段有限
编译框架辅助的指令翻译	LLVM IR[74]	高	强	中等	优化成本较高
规则匹配的指令映射翻译	文献[72, 81-83, 85]等	高	弱	低	指令覆盖率不足;学习成本高;对全系统翻译支持不足.

3 二进制翻译技术关键问题

二进制翻译的输入文件不再维护函数名、变量信息、数据类型等关键符号信息,导致其相比编译器设计更具挑战性.根据表 1 典型二进制翻译系统,我们梳理发现当前二进制翻译技术依旧存在大量热点关键问题.引发这些问题的主要原因有:(1)冯诺依曼结构的指令/地址/数据混编混接、变长指令、间接跳转指令、代码自修改等特性,二进制程序解码和翻译难度大.(2)基于虚拟环境模拟的执行方式,会引入大量翻译和运行开销.(3)不同处理器平台在内存模型、指令集、存储单元等硬件差异,对指令翻译和处理器状态模拟难度较大.

3.1 存储单元模拟

二进制翻译除了完成指令的基本翻译,还需要在目标机器上部署对应的存储单元作为源机器平台中寄存器的镜像,实现源平台的寄存器和内存等存储单元的等价模拟.除了对通用寄存器的镜像部署,程序中专用寄存器的状态同样需要在目标机器中对应部署.例如在翻译 x86 的分段寄存器和标志位寄存器时,在目标机器中也必须建立对应的内存镜像.当前镜像部署的解决方案主要有两种:开辟一块内存空间做镜像映射[13, 16]或者利用硬件支持寄存器映射[17, 53, 87].其中,开辟内存空间的方法屏蔽了寄存器使用差异,实现简单,但会引发大量的访存代价.硬件支持的寄存器映射方法效率较高,但不同处理器平台中寄存器数量和使用约定各异,很难实现寄存器的全部映射.尤其是源平台寄存器数量多于目标平台时,必然有部分源机器平台中的寄存器要被映射到内存单元中,涉及到寄存器映射收益[53]和映射代价选择[83, 88]问题.具体优化方法将在 4.3.1 节中进行讨论.

与存储单元模拟相关的另外一个难点是对内存地址的翻译.系统级翻译时,所有客户机访存操作需要经过两级虚实地址转换,即先把客户机的虚地址翻译成客户机的物理地址.然后再将客户机的物理地址作为宿主机的虚地址,最终翻译成宿主机的物理地址.QEMU 采用软件模拟实现两级虚实地址转换翻译,翻译效率较低.Wang 等人[89]和 Faravelon 等人[90]提出将客户虚拟地址空间嵌入到主机地址空间中,宿主机可无差别地直接访问客户机虚地址.然而,该方法对于客户机和宿主机的虚拟地址空间大小有严格限制.BTMMU[91]通过扩展内存单元部件,在内核模块中增加影子页表来实现内存地址翻译,具有通用性.LAT[92]采用硬件支持的方式实现客户机虚地址到宿主机实地址的直接代换,彻底消除软件模拟虚实地址转换引发的性能开销问题.

3.2 原子操作时序维护

原子操作保证了资源访问的互斥性和访问时序.翻译多节拍原子操作时,需要维持其原来的时序和语义特性.然而,基于软件模拟的指令翻译方式很容易破坏原子操作的原有逻辑时序,引发正确性问题.

原子操作翻译过程中首先面临的一个挑战是原子指令的等价翻译.例如 x86 平台有将近 20 条原子指令,如 inc、xadd 原子操作以及 cmpxchg 比较并交换(compare and swap,CAS)指令,而 MIPS、ARM、Alpha 等平台并没有 CAS 指令,其对应的是链接加载(load linked,LL)和条件存储(store conditional,SC)指令组成的 LL/SC 指令

对.对于 CAS 原子指令的翻译有两种常用的翻译方法:(1)将源程序中所有的内存写操作转为原子写,以保证访问内存数据的一致性.但是这种做法对性能影响大.(2)维持源程序逻辑,利用 CAS 和 LL/SC 指令对进行功能的等价模拟.Kristien 等人[93]研究发现 QEMU 和 ARCSim[94]在利用 CAS 模拟 LL/SC 指令时存在正确性问题,为了保证从 LL/SC 到 CAS 翻译的正确性,提出利用软件模拟页翻译缓存更新和失效机制.Natarajan 等人[95]提出基于内存事务解决多线程应用之间的数据竞争问题,将元数据访问封装在一个事务性原子块内完成,保证了多线程中锁指令翻译的正确性和性能.此外,基于 CAS 模拟 LL/SC 指令还可能会引发 ABA 问题[96].针对 ABA 问题,Rigo 等人[97]提出利用 Helper 函数软件模拟实现 LL/SC 锁指令翻译.Jiang 等人[98]提出无锁队列引用计数的内存保护方法,只有内存引用节点计数值为 0 时才允许访问该内存.Cota 等人[15]提出通过位图维护 Cache 行和哈希完整地址行检查来避免数据竞争并保证数据访问的原子性.Zhao 等人[99]利用插桩的方式维护一个非阻塞哈希表来记录更新内存,保证了锁变量地址与被访问内存地址的一致性.

原子操作翻译过程中还可能会引发锁地址非对齐问题,例如 x86 平台支持多种地址非对齐的原子操作,而 MIPS、ARM、Alpha 等平台要求 LL/SC 指令满足 32 位地址对齐.对于 x86 非对齐指令的翻译通常要求先将访存地址对齐,但在不同位宽的原子指令翻译时依旧可能会引发错误.图 7 是 16 位非对齐指令的访存场景,假设起始地址满足 32 位对齐,其中(1)和(3)是 16 位地址对齐,(2)和(4)是非 16 位地址对齐.在该场景下,用 32 位对齐原子指令模拟 16 位非对齐原子指令,此时(1)(2)(3)可以被同一个 32 位访存指令访问覆盖,而(4)横跨两个 32 位内存地址,翻译可能出错.针对该问题,Jiang 等人[98]利用 LL/SC 指令对模拟 CAS 指令,采用多内存地址比较交换方法解决锁地址非对齐可能引发的错误,但是该方法基于软件模拟 CAS 指令,可能会引发大量的冗余判断.

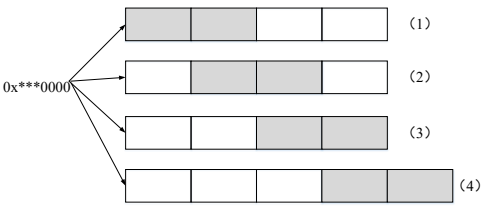


图 7 32 位对齐指令模拟翻译 16 位非对齐指令

3.3 异常和中断处理

代码块执行时遇到异常或者中断,其需要调用回退机制实现源程序状态的精准还原,确保逻辑正确.二进制翻译为了实现回退机制,需要加入额外的保存、恢复机器状态的代码.IA-32 EL[3]采用还原点检测机制,一旦发生异常便从最近的还原点恢复出精确的机器状态.Crusoe[4]采用硬件支持的影子寄存器保存精确状态.然而,回退机制是比较耗时的,研究表明 IA-32 EL 做一次回退机制需要多达 10^4 个时钟周期.此外,考虑到中断是异步事件.QEMU 使用主动处理中断的策略在生成翻译代码阶段插入中断检查代码,避免回退机制产生的开销.Captive[9]使用硬件虚拟化在客户模式下运行一个完整的翻译系统.它的外部中断通过中断控制器传播到客户机系统,并在中断处理程序中设置中断挂起标志.之后,在每个翻译块的末尾检查此标志,如果设置了该标志,则将其清除并执行相关的中断处理程序.但事实上,与执行翻译块相比,中断发生的概率要低得多,翻译器对大多数挂起的中断检查都是非必要的.Niu 等人[100]利用信号和运行时二进制重写技术通知线程何时交付中断,不再重复检查挂起的中断,该方法在实现中断异常检测的同时消除了不必要的开销.

3.4 内存一致性保证

根据处理器对内存模型的支持情况可以分为 3 类:顺序性内存模型(sequentially consistent,SC)、强内存模型(total store order,TSO)和弱内存模型(weak memory order,WMO).表 4 列出了主流处理器架构在硬件层面对并发程序中数据访问顺序一致性的保证情况.可以看出不同处理器硬件对内存模型支持存在很大差异.

表 4 主流处理器硬件对并发程序中存储数据访问顺序一致性保证

执行操作 \ 处理器类型	Alpha	ARMv7	RISC-V		POWER	x86[8]	AMD64
			WMO	TSO			
读后读	×	×	×	√	×	√	√
读后写	×	×	×	√	×	√	√
写后写	×	×	×	√	×	√	√
写后读	×	×	×	×	×	×	×

表 5 给出了同一并发程序在 TSO 和 WMO 两种不同内存模型的处理器中执行的结果.假设初始状态为 A=0,B=0,在场景 1 中,TSO 模型不允许输出 a=0,b=1,而 WMO 模型允许.在场景 2 中,TSO 模型不允许输出 a=0,b=0,而 WMO 模型允许.出现不同输出结果的原因是 TSO 模型通过硬件隐式保证访存的时序性,而 WMO 模型的硬件并没有进行该操作,代码执行时访问数据的顺序可以重排.

表 5 程序在 TSO 和 WMO 内存模型的执行结果

场景	线程 0	线程 1	TSO 内存模型输出		WMO 内存模型输出	
场景 1	A=1 B=1	a=A b=B	(a=0,b=1)	×	(a=0,b=1)	√
场景 2	A=1 b=B	B=1 a=A	(a=0,b=0)	×	(a=0,b=0)	√

TSO 模型通过硬件保证访存的先后顺序,而 WMO 模型的硬件无法保证内存的访问顺序.如果想要将 TSO 模型的应用迁移到 WMO 模型,需要额外加入同步栅栏指令来避免指令发生重排序.然而,在二进制代码中无论是寻找待同步点还是添加栅栏指令均会带来较高的性能开销.针对 TSO-to-WMO 并发程序的翻译,Natarajan 等人[95]量化分析了插入同步栅栏指令和基于事务内存一致性处理两种方法带来的性能开销,发现对于事务冲突较少且事务足够大的并发程序,基于事务的内存一致性处理性能较好.对于事务开销较高的并发程序,插入内存同步栅栏指令的效果更好.然而,相比于使用单一方法,两种方法混合使用效果更优.

栅栏指令的插入会显著影响程序执行效率,选择合适的插入时机就变得极其重要.Chakraborty 等人[101]提出强化弱内存模型变换的鲁棒性,当发现违反鲁棒性时再插入适当的内存栅栏指令.文献[102]在 Coq 工具上设计公理化的宽松内存模型实现内存读写操作的局部重排序,完成从 SC 内存模型向 WMO 内存模型的变换.Lustig 等人[103]提出内存一致性框架 ArMOR,构建内存顺序约束表以确定 load→load、load→store、store→load 以及 store→store 是否需要保序,辅助指导翻译程序动态的注入栅栏同步指令.Emilio 等人[15]证明了插入内存栅栏指令对保证多核程序访存顺序的有效性.Rocha 等人[40]提出的 Lasagne 通过构建语句一致性和访存原子约束条件来保证程序全局访问的内存顺序,扩展 LLVM IR 并发原语确定内存栅栏指令的插入位置,实现由强到弱内存模型翻译的 LIMM(LLVM IR Memory Model)并发模型,之后基于 LIMM 模型实现从 x86 平台到 LLVM IR 再到 ARM 平台的转换.Lasagne 很好解决了静态翻译在不同内存模型的硬件平台之间正确翻译并发程序难题.QEMU 引入多线程 MTTCG(multi-threaded TCG)来支持不同内存一致性模型并发程序的翻译.最近,Gouicem 等人[61]研究发现 QEMU 对并发程序的翻译存在多个翻译错误,对此,Gouicem 等人提出了名为 Risotto 的并发程序翻译方法.Risotto 形式化了基于 TCG IR 中间表示的内存模型转换过程,并使用被形式化的 TCG IR 实现强内存模型到弱内存模型的内存映射.Risotto 很好解决了 QEMU 在不同内存模型的硬件平台之间正确翻译并发程序的难题.

3.5 代码挖掘

与代码挖掘相关的两个问题包括代码自修改和自生成.代码自修改在 Adobe Premiere、Doom 以及嵌入式应用中较为常见,代码自生成则随着 JavaScript、PHP、C#等脚本语言的流行变得更加普遍[104].为了保证缓存代码的及时更新,当发生代码自修改和代码自生成时,二进制翻译需要满足:(1)及时发现被修改代码块.(2)重新翻译被修改代码并更新缓存.目前,静态翻译无法预知代码自修改和自生成,动态翻译虽然能够及时发现变化的代

[8] 部分旧的 x86 和 AMD 处理器采用弱内存模型

码,但是在代码块的发现和缓存更新方面开销较大.

在修改代码块的发现上,动态翻译主要是基于信号探测和代码注释的方法实现.QEMU 采用 mprotect 方法将所有代码设置为不可写,通过监测 SIGSEGV 异常来发现自修改情况.Strata[105]基于内存读写保护和信号探测来发现自修改行为.Liu 等人[106]通过比较源代码和备份源代码的异同,进而确定代码是否被修改.Hawkins 等人[104]基于代码注释的方式来发现自生成的代码区域;当代码发生变化时,Transmeta CMS[4]、DAISY[6]、QEMU 等丢弃所有受影响的缓存代码,开展重新翻译.这种做法可能会导致大量未修改代码被淘汰和重复翻译.文献[106]设计精确化自修改代码更新机制,如果修改后的基本块小于原基本块,则直接使用修改后的基本块替换原基本块.如果修改后的基本块大于原基本块,则为修改后的基本块新开辟代码空间.Wang 等人[107]提出最大化复用基本块代码,发生页故障时,启动代码备份机制保存该页代码到新开辟的一个代码空间,然后逐项比较被修改源代码块与备份块,仅在实际修改源代码块时才会重新翻译代码块.

冯·诺伊曼结构的机器中代码和数据以相同形式表示,重定位、指令跳转、空指令填充等特征导致代码边界模糊,难以区分代码段和数据段.静态翻译自身的局限性导致其在自修改代码的发现上存在很大挑战.最常用的代码发现方法是从可执行文件的执行入口处逐条解码挖掘文件中的可执行指令,但是间接转移这类指令的存在增加了静态翻译中代码发现的难度.Cifuentes 等人[108]提出基于分割和表达式替换方法恢复跳转表目标地址的技术,改善了静态翻译对代码发现不足的问题.Chen 等人[109]提出采用多轮线性扫描的方法明确代码边界,精确区分数据段和代码段的自修改情况.但是在发生自修改之后需要对缓存代码及时更新,这对于静态翻译来说依旧是个难题.

3.6 功能等价性研究

二进制翻译过程是软件功能在新平台环境的再实现,翻译过程以源程序逻辑等价性为前提.逻辑的等价性包括翻译前后程序行为的一致性、性能表现的等效性等.然而,逻辑等价性验证面临着大量挑战:(1)二进制翻译技术的跨平台特性导致大量测试集依赖于处理器架构[110],不同平台之间的回归测试集难以直接复用[111].(2)二进制翻译系统对自身的代码改动极其敏感,开发过程中需要频繁的引入测试.然而,当前业界普遍采用 SPEC2006、SPEC2017、LTP 等大型测试集进行测试,测试过程十分耗时[112].(3)二进制翻译系统的功能设计复杂,测试过程中很难保证对于功能点的全覆盖.研究发现即使采用大型测试集进行测试,依旧有 30%的二进制翻译系统代码功能点无法被覆盖[113].(4)等价性测试主要是基于特定测试集的白盒与黑盒测试,而要验证翻译程序的功能等价性是极具困难的.我们将二进制翻译的功能等价性研究归纳为软件测试和验证研究.

在软件测试研究中,为了提高翻译代码功能点的覆盖率,Guo 等人[113]提出基于翻译平台指令和操作数随机生成测试用例的方法,有效提升了测试代码覆盖率.Zhi 等人[111]提出基于 LLVM IR 中间表示变换将程序统一转换为 x86 平台二进制代码,进而复用 x86 现有的程序测试工具,弥补了其他目标平台测试工具匮乏的不足.Wu 等人[114]提出基于 PerfDBT 构建小型回归测试集,运行分析已有测试集的基本块,将热路径代码规模缩减后形成新的基准回归测试集,有效提升了已有回归测试集的使用效率.Wu 等人[112]提出了 FADATest 框架,首先利用动态二进制翻译执行已有标准测试集程序并捕获测试程序运行特征信息,然后基于此特征信息生成新的测试代码,用以模拟原始基准程序行为.FADATest 弥补了 PerfDBT[114]测试集在跨平台和多线程应用方面支持的不足缺陷.此外,Wagstaff 等人[115]提出支持跨平台全系统模拟的基准测试 SimBench,支持全系统翻译中的性能评估、中断和异常处理、内存访问、I/O 以及其他性能敏感的代码测试.

在验证研究中,Kim 等人[116]基于符号执行方法将多个不同版本的代码全部提升至 IR,通过比较 IR 语义差异,找出代码翻译过程中可能存在的错误,但是该方法不支持浮点运算指令的语义等价性验证.Chen 等人[117]同时执行翻译前和翻译后的代码,逐个对比二者的执行过程状态变化和内存操作行为,以确定翻译前后代码语义的等价性.Koltunov 等人[118]以同一份代码在不同平台上的运行行为一致为前提,将 C 语言代码编译后分别在目标平台和 QEMU 上模拟运行,通过比较两个版本的代码控制流和运行输出来检测 TCG IR 的语义正确性.Sandeep 等人[119]使用符号执行和定理证明方法验证单条指令翻译前后的等价性,然后基于经过验证的指令形成参考标准语义.最后基于图同构检查方法验证翻译后的代码与参考标准语义的等价性.Fu 等人[120]通过形

式化验证方法构建能够涵盖动态二进制翻译和静态二进制翻译的统一抽象模型,为二进制翻译的功能扩展和性能提升研究提供了理论支撑,但其并未就该抽象模型对具体的二进制翻译器进行实践证明.

此外,针对二进制翻译过程对应用程序原有安全防护功能的影响,Chen 等人[121]逐项分析二进制翻译对缓冲区溢出防护、恶意代码攻击检测、系统调用防护、防止逆向和安全断言检查功能的影响,总结如表 6 所示.研究发现,对于软件静态防护、加载时防护以及由软件自身发起的运行时防护措施,原有安全功能不受二进制翻译过程影响,而依赖第三方应用的安全防护以及二进制翻译系统无法识别的安全防护在翻译过程中会被禁用或失效.

表 6 二进制翻译对软件安全防护措施功能影响

软件安全防护		基于二进制翻译运行
缓冲区溢出防护	Stackshield[122]	有效
	Propolice 和 Stackguard[123]	有效
	Libsafe[124]	有效
	地址随机化[125]	有效
恶意代码攻击检测	自校验[126]	在自引用属性保证前提下有效
	水印技术[127]	有效
系统调用防护	系统调用沙箱[128]	有效
	系统调用认证[129]	失效
防止逆向	代码混淆和代码变形[130]	部分有效
	反调试[131]	除基于时间方法之外有效
	指令随机化[132]	部分有效
安全断言检查	携带证明代码[133]	有效

3.7 翻译效率提升

翻译效率是度量二进制翻译系统性能优劣最常用的指标[12, 17, 37].翻译效率越高,则表明二进制翻译系统越优秀.因此,翻译效率的提升是业界持续关注的热点问题.当前成熟的二进制翻译系统在翻译效率上取得了显著提升,例如,基于 SPEC2006 测试集测试,ExaGear 实现 x86-to-ARM64 的平均翻译效率达 82%[12];Tango 实现 ARM32-to-ARM64 的平均翻译效率达 80%[37];Rosetta 2 实现 x86-64-to-ARM64 的平均翻译效率超过 75%[5];LAT 实现 x86-to-LoongArch 的平均翻译效率超过 65%[17].

本文将二进制翻译效率计算公式定义如下:

$$翻译效率=\frac{本地原生代码执行时间}{基于二进制翻译执行时间}*100\% \tag{1}$$

其中,本地原生代码执行是指在目标平台上直接运行原生编译的二进制代码;基于二进制翻译执行是指在目标平台上利用二进制翻译技术模拟执行源程序的过程,涉及开销包括指令翻译、运行时维护和目标代码执行三部分.其中,翻译开销包括对指令的翻译、程序状态模拟、运行环境准备等开销.运行时开销包括控制流切换、缓存代码查找、跳转目标地址计算等开销.目标代码执行开销与本地原生代码执行类似.理想情况下,目标代码执行越接近本地原生代码执行效果,表示翻译生成的代码质量越高.

然而,提升翻译效率并非易事,主要原因有:(1)二进制翻译系统自身在指令翻译、代码优化中引发了大量开销;(2)不同硬件平台存在差异,为实现指令等价翻译和机器状态正确模拟,二进制翻译时会引入大量冗余操作.(3)基于虚拟环境模拟的方法,二进制翻译涉及缓存代码管理、分支跳转地址计算、控制流切换等运行时开销.(4)翻译过程需精准维护源程序逻辑,目标平台处理器硬件资源的利用会受到源程序特征的影响.如目标平台在多核资源的利用上不能改变源程序设置的最大处理器核数.

为了提升翻译效率,当前已有大量优化研究工作.例如降低翻译开销的动静结合[55, 134, 135]、多线程优化[36] [15, 62];减少运行时维护开销的分支查找优化[23, 27, 136]、热路径生成[20, 137, 138];提升目标代码生成质量的中间代码优化[138, 139]、指令并行化[136, 140-142]、寄存器映射[53, 83, 136, 143, 144];缩小源平台和目标平台体系结构差异的软硬协同设计[10, 145]高级优化.相关优化工作将在第 4 节做进一步介绍.

3.8 小结

本节梳理了当前二进制翻译技术面临的核心关键问题,包括存储单元模拟、原子指令时序特征维护、异常和中断处理、内存一致性保证、代码挖掘、等价性验证、翻译效率提升等.此外,全系统翻译还需要完成系统指令和特权级的正确模拟,但是该部分隶属于虚拟化技术范畴,本文不做重点讨论.

总结发现,当前研究针对关键问题成果显著,均提供了行之有效的解决策略.但是已有研究主要聚焦于特定架构或应用场景暴露出来的问题,一些潜在的研究方向仍有待进一步探索,例如基于形式化证明二进制翻译的功能等价性等.

4 二进制翻译技术性能优化

翻译效率的高低作为制约二进制翻译技术发展的关键因素,直接决定着二进制翻译系统的实用性.然而,基于虚拟环境模拟的二进制翻译技术的本质决定了其在翻译效率上很难达到本地原生运行效果.为保证不同平台之间软件的平滑迁移和运行,持续开展二进制翻译的性能优化始终是业界的研究热点.

4.1 翻译开销优化

一款好的二进制翻译系统应该既能精确模拟程序行为,又能生成接近或者超过本地原生编译质量的目标代码,同时还要避免翻译系统自身带来的性能开销.

4.1.1 动静结合翻译优化

静态翻译效率较高,但在间接跳转、代码自修改、代码自生成等情况处理上存在不足,在实际应用时受到诸多限制.动态翻译虽然能够解决静态翻译存在的不足,在实际应用中适用性更强,但是有着翻译开销大和代码优化不充分等不足,翻译效率较低.将静态翻译和动态翻译相结合的动静结合翻译,既准确获取程序的运行时信息保证翻译的完整性,又可以降低翻译开销.动静结合的设计方法已被成熟的翻译系统普遍使用.总结起来,动静结合的翻译方法有静态为主动态为辅和动态为主静态为辅两种形式.

静态为主动态为辅是以静态翻译框架为主体,结合动态翻译器提供的运行时信息,解决了静态翻译对程序运行时信息无法获知的难题,又有效避免了翻译开销的引入.Rosetta 2[5]引入 AOT 和 JIT 技术实现动静结合翻译,预先翻译对性能潜在影响较大的代码,并将预先翻译代码以映像文件的形式写入存储空间.对于参数未知或者需要运行时解析的场景,则使用 JIT 实时翻译执行.Shen 等人[135]利用静态翻译尽可能提前翻译应用程序,然后在执行预先翻译代码时,如果遇到间接跳转等不确定情况再调用动态翻译进行补充翻译.实验表明该方法相比完全使用动态翻译性能平均提升 8 倍.文献[146]提出基于 LLVM IR 的 Rabbit 动静结合翻译框架,相比文献[135]在静态翻译的分支跳转方面有所突破.Rabbit 基于解释执行非热点函数,并缓存解释执行的指令信息供再次执行时复用.Sun 等人[147]针对静态翻译中间接跳转无法提前获知的问题,提出一种基于分支预测的动静结合二进制翻译框架 BP-QEMU.BP-QEMU 首先基于静态翻译预先翻译二进制程序,在翻译过程中过滤分支跳转代码模块并进行跳转目标预测.然后使用动态翻译处理静态翻译阶段未处理的翻译块,并确认静态翻译预测的跳转目标地址的正确性.相比于 Rabbit,BP-QEMU 利用静态翻译对代码预先翻译方面有更大的突破.

动态为主静态为辅是以动态翻译框架为主体,利用静态翻译充分离线优化翻译后代码,解决了动态翻译开销和性能优化开销大的难题.Hu 等人[10]基于动态翻译基本块并采样分析热点函数,对发现的热点函数调用超级块优化器进行离线优化.类似地,MTCrossBit[36]将代码优化工作与翻译工作剥离,在首次动态翻译时额外插入探测指令收集程序的运行时信息,之后根据收集到的信息指导静态翻译完成离线优化.为进一步减少运行时收集信息带来的开销,Guan 等人[55]在 MTCrossBit 的基础上,提出只收集跟踪执行路径上相邻边的执行信息,大幅降低了执行信息的数据规模.此外,Wang 等人[148]提出对代码进行静态预翻译并完成深入优化,将优化后的代码以动态库的形式提供给动态翻译调用.动态翻译主要负责程序内存空间映射、程序状态更新以及共享库调用执行等任务.LLPEMU[34]采用了与文献[148]类似的方法,引入多面体优化技术对静态翻译发现的程序循环体进行深度优化并提供给动态翻译调用.

4.1.2 多线程翻译优化

动静结合的优化方法有效降低了代码翻译和优化开销,但是其对于处理器的多核资源利用还不够.为此,有研究利用多线程技术优化翻译开销.传统的二进制翻译系统采用单线程模式:串行处理代码翻译和代码执行任务,导致大量资源互相等待,处理器资源利用率低下.为了解决该问题,一些研究工作提出充分利用多核处理器体系结构的并行优势,将翻译和执行任务划分到不同线程,尽可能的隐藏翻译过程带来的开销.多线程翻译在任务划分上可以分为一对一翻译、一对多翻译以及主从翻译 3 种模式.

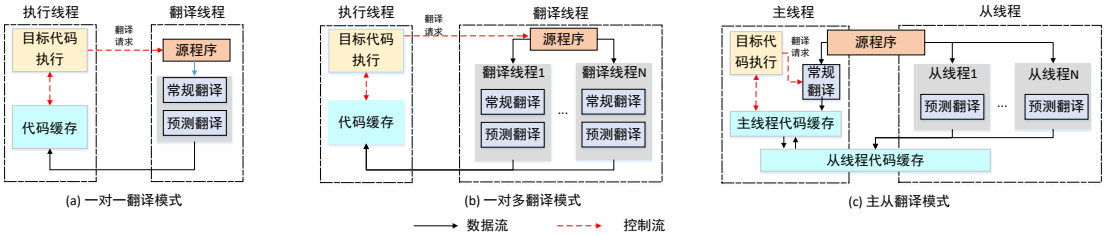


图 8 二进制翻译中多线程翻译模式(“彩印”)

(1)一对一翻译模式

一对一翻译模式包含一个执行线程和一个翻译线程,翻译线程和执行线程的并行化保证了在代码执行时可以同步进行代码翻译,如图 8(a)所示.翻译线程负责对源程序进行翻译,包括常规翻译和预测翻译.常规翻译负责对执行过程中未被翻译的源程序进行翻译,预测翻译则对可能被执行的目标代码进行预测翻译.执行线程负责执行翻译后的目标代码.相比于单线程翻译,一对一的多线程翻译模式提高了二进制翻译的并行度,但是也存在一些缺陷:当执行线程提出翻译请求时,预测翻译无法同时进行,此时就退化成了单线程模式.此外,单个基本块的翻译时间可能比执行时间长,当翻译线程处于预测翻译时,执行线程发出的新的翻译请求无法被及时响应,严重影响整体翻译效率.多线程 Strata[22]采用了一对一翻译模式.

(2)一对多翻译模式

相比一对一翻译模式,一对多翻译模式支持动态扩充翻译线程数量[20],如图 8(b)所示.为了充分提升线程的并行处理能力,Ma 等人[149]提出创建翻译线程、超级块线程、性能剖析线程以及执行线程的多线程翻译系统,通过高效的利用多核资源提升系统的整体性能.HQEMU[20]将 QEMU TCG 翻译器和 LLVM 优化器分别运行在不同线程上,根据优化任务量动态的调整优化器线程数量.Liu 等人[150]基于离线采样策略构建 CFG 实现预测翻译,以流水线的方式运行前端到 IR 提升、IR 到后端翻译和代码执行.Engelke 等人[75]构建了客户端+服务端的多进程翻译框架,根据任务需求创建多线程,有效的解决了翻译请求无法及时响应的问题.类似地,ExaGear[12]采用了两级翻译模式,一级进程负责快速翻译,二级优化进程则对热点代码进行深层次并行优化,最终将优化后代码重新注入到代码池供后续程序执行使用.一对多翻译模式解决了一对一翻译模式并发度低和响应翻译请求不及时的缺陷,可以在同一时间响应多个翻译任务.但是当所有线程均在执行预测翻译时,一对多翻译模式也面临着无法及时响应执行线程发出的翻译请求的可能性.

(3)主从翻译模式

一对一翻译模式和一对多翻译模式将源程序的代码翻译与执行过程完全分离,翻译线程和执行线程无优先级区分,可能存在线程之间任务请求无法及时响应的问题.基于主从结构的多线程翻译模式解决了该问题.主从翻译模式将线程分为主线程和从线程两种类型,主线程负责常规翻译和目标代码执行,从线程负责对未翻译的代码进行预测翻译,执行流程如图 8(c)所示.在任务分派时,首先将待翻译任务加入循环队列,从线程在队列中领取翻译任务执行预测翻译.在目标代码执行时发出的翻译请求则由主线程以最快的响应速度直接翻译.主线程和从线程分别维护各自的代码缓存,并且主线程可以访问所有从线程的缓存.MT-BTRIMER[151]是典型的主从模式多线程翻译器,MT-BTRIMER 在翻译效率上优于单线程模式,为二进制翻译器在多线程性能优化方面提供了一个高效灵活的框架.

4.2 运行时优化

动态优化在程序运行时采集程序的运行信息,作为进一步优化的依据.动态优化可以进一步挖掘静态时无法发现的优化机会,是程序性能优化的一种有效方法.二进制翻译执行过程涉及分支跳转目标地址计算、缓存代码管理、程序状态信息维护等工作,期间需要频繁的控制流切换和上下文保存恢复,显著增加了运行时维护开销.因此,有必要开展相关优化工作来降低运行时开销.

4.2.1 热路径优化

对大部分程序而言,20%的代码占据了 80%以上的运行时间[152],充分优化频繁执行的热路径代码可以减少基本块之间的切换和代码块查找开销,提高程序的整体性能.其中热路径优化作为一项动态优化技术,其可以减少基本块之间的切换和代码块查找开销.运行时采样和代码插桩的方法是现阶段热路径发现的常用手段.运行时采样通过 CPU 内置计数器、触发中断或者直接解释执行来完成,例如 Aries[2]、BOA[7]、Walkabout[47]、ARCSim[52]和 Rv8[58]代码插桩通过在基本块和控制流边缘插入监控代码来识别程序热点.代码插桩通过在基本块和控制流边缘插入监控代码来识别程序热点,例如 IA-32EL[3]和 HQEMU[20]基于此方法.

构建更大的代码块区域是热路径优化的一种有效手段,该方法使得每个基本块执行结束后直接跳入后继基本块继续执行,减少了寄存器保留恢复与控制流切换次数.文献[153]借助处理器硬件收集程序执行信息,并基于有向图构建来表示执行节点或边的频率,丰富历史分支记录信息以构造高质量区域代码.代码块的区域构建扩大了代码块区域,暴露出更多的优化机会.HQEMU[20]将热路径代码的基本块组成 Trace 链,并发送给 LLVM 优化器重新翻译优化,引入 Trace 链合并后部分课题性能提升了 71%.文献[137]充分延长 Trace 链,对代码块区域开展删除不常用代码、增加代码局部性、改进传统优化和增加推测等优化.Ispike[138]将执行频率高的基本块组成超级链以提高 Cache 利用率,并且分离出函数中的热路径代码和冷执行代码,减少控制流转换开销.

多线程中目标程序的代码块信息是线程私有的,因此在线程共享机制下创建和保存 Trace 链是具有挑战性的工作.DynamoRIO[42]为保证线程同步,当有线程开始创建 Trace 链时其会同时设置标志来阻止其他线程重复创建 Trace 链,同时其还采用了线程私有的数据结构保存 Trace 链,在安全点再保存到共享代码缓存中.

4.2.2 分支跳转优化

分支跳转包括直接分支跳转和间接分支跳转两类.直接分支跳转的目标地址唯一,通过程序运行时地址回填等方法即可确定.间接分支跳转的目标地址在程序执行时确定,跳转目标地址不唯一.对于间接分支跳转目标地址的确定,通常是首先从缓冲区代码中进行查找,如果查找失败则控制流切换到翻译器开展翻译.此过程涉及到频繁的控制流切换和翻译查找.间接分支跳转目标地址的计算是影响二进制翻译效率的关键瓶颈[23, 27, 136].

间接分支跳转目标地址查找是确定源程序跳转地址和目标程序跳转地址关系映射的过程,查找十分耗时.D'Antras 等人[139]基于硬件辅助生成一个包含多 case 目标地址的分支跳转表,通过目标地址反推理法获取间接分支跳转指令,引入间接跳转优化后,可以降低 40%的运行时维护开销.对于未涵盖的其余间接分支使用快速原子哈希表处理,将每个哈希表条目的源和目标地址打包为 64 位指针并原子的从共享缓存中读写,避免在分支查询时加入栏栅同步指令,加速多线程翻译对间接跳转的同步效率.Shen 等人[54]构建 ARM-to-LLVM IR 的地址映射表来加速间接分支跳转,为了避免映射表过于庞大,该方法只处理函数入口点、返回地址、函数指针地址以及虚函数四种间接跳转.LAT[17]为每一个包含间接分支的基本块增加一个私有缓存以保存分支目标地址,提高查找效率.Huang 等人[154]在每个基本块的头部插入比较跳转指令,将间接跳转目标地址预测转移到基本块内部完成.

为了降低间接分支跳转产生的控制流切换开销,Pin[32]使用间接分支链将所有间接分支指令关联起来,将间接跳转转化为间接目标地址移动和预测目标地址的直接跳转.Dynamorio[42]通过查找哈希表的方法将间接跳转连接起来.相比于 Dynamorio,Pin 采用的间接链接机制更加灵活和高效.文献[155]在构建 Trace 时内联间接分支,将间接跳转比较运算转移到 Trace 块内完成.Chen 等人[156]将所有包含间接跳转操作的基本块合并成一个超级块并在超级块内完成分支跳转操作.

针对静态翻译无法确定间接分支跳转目标地址的难题,Kinder 等人[157]提出基于数据流分析重构出近似完

整的程序控制流图,获取程序的分支跳转目标地址.Wang 等人[158]提出在解析间接跳转指令时加入异常处理指令.异常探测方法用于确定目标跳转地址并回填,执行多轮探测直至完成所有目标地址的回填.类似地,Federico等人[159]提出在静态翻译时预收集程序全局数据的基础上,引入表达式跟踪与数据流范围分析来确定基本块边界,从而确定跳转目的地址.这些研究在静态翻译阶段确定间接分支跳转目标地址的问题上取得了重要突破.

4.2.3 代码缓存优化

缓存已翻译代码可有效避免重复翻译,代码缓存设计是当前二进制翻译系统的普遍做法[21, 134, 160, 161].在二进制翻译执行过程中,当不确定后续执行代码块时,优先从缓存中查找,如果未命中再切换到翻译器开展翻译.对于缓存的查找也会带来额外的代码查找开销.Chen 等人[134]改进了解释执行的代码处理方式,对解释执行的代码进行缓存,当程序再次执行时优先从缓存中查询,提升了基于解释执行方法的翻译效率.代码缓存方法降低了指令翻译翻译,但是也会带来额外的代码查找开销.QEMU 采用“快速查找+慢速查找”的两级缓存查找方法,优先基于快速查找从热路径代码组成的缓存空间中查找,如未命中再切换到慢速查找对所有缓存进行遍历,效率较低.Yue 等人[21]改进了 QEMU 在两级缓存查找时采用的缓存置换策略,以页为单位对缓存代码进行划分和热度标记,减少了缓存替换颠簸.Pico[15]优化了 QEMU 中翻译块缓存和相关查找机制,改进哈希函数并构建新的哈希表,提升了缓存代码的查找效率.

对于运行时间短或者包含大量冷代码的程序来说,代码缓存摊销翻译成本的效果十分有限.对于无明显热路径代码的应用,冷代码翻译与执行开销也是影响整个系统性能的关键[160].Wang 等人[160]研究发现在 GUI 应用中,90%的翻译时间花费在冷代码处理上.为了进一步提高冷代码翻译效率,Wang 等人[161]提出持久性常驻缓存代码的方法,允许相同或不同的应用程序在执行时复用同一份缓存代码,进一步减少了翻译过程带来的开销.

此外,对于多线程应用翻译,存在缓存资源访问竞争和数据同步开销问题.针对此问题,Hong 等人[155]提出私有化每个执行线程的缓存表来避免数据访问竞争.对于多线程中缓存同步策略的优化,COREMU[28]采用延迟失效策略解决多线程页失效的同步问题,减少了多线程之间的数据同步开销.

4.3 代码生成优化

翻译开销优化和运行时优化有效降低了二进制翻译器带来的开销.然而,翻译后代码作为本地重复执行的的目标程序,对其更深层次的优化可以进一步提升二进制翻译效率.本文基于 QEMU 完成 x86-to-SW64 的翻译,统计 SPEC2006 测试集执行时间,发现目标代码执行时间占比达 95%以上.此外,Wang 等人[161]研究发现对于计算密集型的应用程序,二进制翻译过程大约有 90%的时间开销花费在目标代码执行上,由此可见生成高质量的目标代码对提升翻译效率至关重要.然而,源平台和目标平台之间在寄存器约定、存储单元、指令集功能等方面存在较大差异,反映到目标代码上,则存在访问内存频繁、代码膨胀率高、指令模拟低效等不足.利用目标平台特性来改进目标代码生成质量,从而缓解体系结构差异带来的低效翻译问题.

4.3.1 寄存器优化

Amr 等人[162]统计 Windows 7 操作系统中指令对寄存器的访问需求,发现超过 75%的指令执行需要访问寄存器,可见有效提升寄存器利用率可以提升二进制翻译效率.在指令翻译时,研究者致力研究各种寄存器单元镜像映射设置策略,如表 7 给出了二进制翻译中常用寄存器镜像映射策略和各自的优劣势.

表 7 二进制翻译中常用的寄存器映射策略

寄存器镜像映射	映射手段	优势	劣势	代表性的翻译系统
虚拟内存映射	使用一片连续内存区域模拟寄存器,在目标平台重新分配寄存器访问内存区域	消除不同平台间寄存器的使用差异	引发大量的访存操作	QEMU[13]
局部寄存器映射	在区域内直接映射源和目标平台物理寄存器.局部区域入口和出口处保存与恢复被映射的寄存器	减少了内存访问操作.实现较为灵活	对于基本块较短的控制密集型程序效果不理想	Bintrans[46] Harmonia[53]
寄存器全局映射	对源程序中涉及到的寄存器做源平台和目标平台寄存器全局映射	引入中间变量较少.减少了内存访问操作	寄存器被定向使用,容易引发寄存器访问冲突	HBT[135]

寄存器映射方法有效缓解了访存压力,然而,源平台和目标平台的物理寄存器在数量、使用方法上存在较大差异,很难有一种寄存器映射策略满足翻译系统的全部设计需求.针对源平台寄存器数量多于目标平台的情况,文献[136]提出将目标平台向量寄存器的低位与源平台寄存器进行映射,缓解目标平台寄存器数量不足的问题.此外,一些研究者提出根据不同算法确定寄存器优先级,以保证寄存器资源的利用率.文献[163]对基本块内中间指令的寄存器需求次数进行排序,优先为排序靠前的变量分配寄存器.文献[53]和文献[83]对代码映射区域进行访存频率排序,提出优先对引发高频率访存的代码区域执行寄存器映射,保证了寄存器映射资源的利用率最大化.文献[164]根据程序对不同寄存器的使用频率确定寄存器分配需求优先级,并在寄存器分配过程中根据优先级动态的调整分配顺序.此外,Fu 等人[88]引入启发式寄存器匹配算法和用于性能上界估计的穷举搜索算法来指导寄存器的优先级,选择最佳的寄存器映射配置.

除了在源平台与目标平台之间进行部分寄存器映射,二进制翻译在生成目标代码时还涉及到目标平台生成代码的物理寄存器分配.一种优秀的寄存器映射策略应该尽可能的将程序变量保存在寄存器中,减少对内存的访问.为了提高二进制翻译中寄存器分配效率,文献[143]在 QEMU 后端增加了线性扫描寄存器分配算法,很好地权衡了目标平台的寄存器分配效果与分配效率之间的关系.Wen 等人[144]提出分段映射和特殊寄存器功能剪裁相结合的寄存器分配方法,提升了寄存器使用的灵活性.

4.3.2 中间代码优化

基于中间表示转换的指令翻译方法降低了翻译难度,同时也引入了大量的冗余指令.通过对中间表示开展优化,可以有效降低代码膨胀率.Li 等人[18]提出基于模式匹配算法替换中间次优代码段.文献[66]通过构建函数控制流与合并基本块,为循环展开、构造跟踪和函数内联提供优化机会.文献[165]将源平台内存地址空间映射到翻译器的同一地址空间,减少了访存查询和地址代换的指令生成开销.文献[56]挖掘目标平台指令之间的数据依赖关系,基于数据依赖图优化算术运算、访存和函数调用等冗余指令.Wu 等人[166]引入标志位指示来标识源平台寄存器的模拟方式,减少二进制翻译过程中基于内存模拟的指令数量.除了传统的优化算法外,也有研究致力于将编译器优化技术应用于二进制翻译中.文献[34]利用编译器的多面体优化来优化中间代码,为自动向量化和热点卸载等优化方向提供了可能.然而,中间代码优化属于一种局部性优化策略,总体来看,在翻译效率提升上,该优化很难有一个数量级的提升.

4.3.3 指令并行化

随着多核处理器的普及与并行编程技术的发展,利用二进制翻译技术发挥目标平台的硬件算力来提升数据并行性成为新的热点.

(1)向量指令翻译重组

向量指令强大的并行能力可以降低冗余指令获取、解码、数据依赖检查和结果写回等开销,当前已在各类处理器中得到了广泛支持,例如 x86 平台的 SSE/AVX、ARM 平台的 NEON、PowerPC 平台的 AltiVec、SW64 和 MIPS 平台的 SIMD 指令等.向量指令在多媒体、2D/3D 图像和游戏应用中广泛使用.Wu 等人[136]统计了 SPEC2017、Gedit、GoogleV8、Totem、VisualStudio 等 9 款 x86 典型应用中的向量指令,发现向量指令在总指令数的平均占比约为 3.1%.可见当前应用对向量指令的支持具有普遍性.

由于向量指令自身设计的复杂性,二进制翻译对向量指令的翻译通常是采用低效的标量指令或者 Helper 函数模拟实现,例如 QEMU 和 HQEMU 默认采用低效的 Helper 函数模拟向量指令,导致翻译效率大打折扣.文献[25]通过重写 Helper 函数和添加向量 TCG IR 来改进 HQEMU 对向量指令的翻译支持,借助 LLVM IR 对向量指令的描述支持,最终在目标平台生成更加高效的向量指令,测试程序性能加速比可达 2.03 倍.考虑到一些 SIMD 指令的语义翻译复杂性,文献[11]优化了文献[25]中的方法,提出混合使用向量 TCG IR 和 Helper 函数向量重写的翻译方法,使得 HQEMU 对向量指令的翻译重组更加灵活.针对结构体存取向量指令翻译效率较低的问题,Fu 等人[88]提出基于连续向量访存加重组操作指令的翻译方法,使用数据重组指令重组结构化数据元素,实现从源平台到目标平台的向量指令生成.

针对翻译过程中源平台和目标平台的向量指令位宽不对称问题,Hallou 等人[167]完成了 x86 同平台短向量

向长向量的映射,实现源程序的 SSE 指令到目标程序的 AVX 指令的翻译.Liu 等人[140]提出了合并短向量指令来生成长向量指令的算法,从而充分利用目标平台并行性并减少寄存器溢出.然而,源平台和目标平台对向量指令的位宽支持存在差异,二进制翻译在对源程序进行循环控制剥离和向量宽度拓宽时可能会引发地址不对齐问题.图 9 中 S0-S10 为一段向量指令,其中 S4-S10 为一段可以向量化的循环指令.二进制翻译在首次翻译执行 S0-S10 代码段时,确定内层 S4-S10 可以向量化.此时由于 S0-S10 执行时一并执行了内层 S4-S10 循环,循环指令的内存起始地址被右移,本来已经对齐的内存地址可能不再对齐.文献[141]提出动态剥离循环代码,当内存起始地址满足长字宽向量指令的对齐要求后再生成长向量指令,该方法使用投票算法计算剥离计数,确保尽可能多的长向量对齐.Hong 等人[142]在文献[141]的基础上借鉴编译器制导标记方法计算程序内存引用依赖距离,优化代码剥离过程中不必要的内存依赖分析.

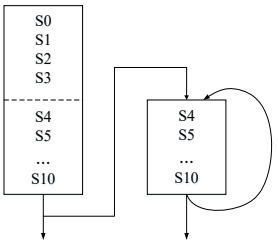


图 9 向量翻译优化导致地址不对齐的场景

(2)标量指令并行化

大量遗留应用程序在最初编译构建时基于标量指令实现,并未充分利用处理器并行性.例如,早期 RISC-V 处理器没有向量扩展指令支持,无法向量化程序循环.当支持向量单元的新 RISC-V 处理器推出时,原来仅用标量实现的遗留二进制程序可以通过二进制翻译优化生成支持向量指令的代码.Nakamura 等人[168]基于程序控制流和数据流信息构建依赖关系图,挖掘指令数据并行性以生成向量指令,实验表明该方法与编译器自动向量化的优化效果相近,但该方法只支持简单数据流向量化.Lin 等人[169]利用虚拟寄存器恢复标量循环关键信息,实现循环内标量指令向量化,该方法支持了单出口的内层循环场景向量化.Zhou 等人[170]提出基于并行规则提取的指令向量化方法,采用动静结合的方法匹配满足翻译规则的代码段,实现遗留应用程序的并行化.Wu 等人[136]对寄存器映射过程中使用向量寄存器低位的标量指令进行特征分析,将满足向量化特征的标量指令直接翻译生成向量指令.Kengo 等人[171]提出一种基于 LLVM IR 转换的串行程序自动并行优化方法,提升二进制代码至 LLVM IR,识别出可并行代码并插入 Openmp 接口,实验表明该方法可以达到与使用源代码并行化同水平的加速效果.

4.3.4 特殊指令优化

跨平台指令集的差异性是限制二进制翻译生成高效目标码的一个关键瓶颈,对于体系结构差异性较大的特殊指令,可以采用解释执行或者高级语言实现的函数模拟,然而这种翻译方法的翻译效率差强人意.

不同平台的浮点和向量指令在精度处理、异常舍入等方面存在差异,为了能精确模拟浮点结果,QEMU 采用高级语言实现的 Helper 函数模拟,并未考虑目标平台硬件特性,翻译效率较低.Shi 等人[26]提出了充分利用目标平台硬件浮点特性来简化浮点 Helper 函数模拟机制,减少了不必要的边界判断,但该方法对浮点异常处理不足.Cota 等人[172]提出浮点运算部分采用硬件浮点指令实现,异常处理部分则调用 Helper 函数模拟实现.该方法既保证了二进制翻译对浮点指令模拟的正确性,又充分利用了目标平台硬件特性.针对 Helper 函数引发的函数调用开销问题,Wang 等人[27]提出将 Helper 函数全部内联到翻译代码中,减少不必要的函数调用和上下文切换开销.Guo 等人[173]提出将浮点指令全部转换成 LLVM IR 表示,不再依赖 Helper 函数,然后利用 LLVM 编译器实现目标平台硬件支持的浮点指令生成.

对于携带标志位指令的模拟是影响二进制翻译性能的又一个关键瓶颈.x86 和 ARM 等架构采用专用标志位处理条件转移指令,并基于标志位实时反映处理器运行时的各种状态和运行结果.而 MIPS、RISC-V、Alpha

等并没有标志位寄存器,翻译时采用软件模拟标志位运算.标志位的模拟不仅要完成标志寄存器逻辑功能,还要实时更新计算标志位的状态变化,引入了大量的内存访问和额外计算开销.文献[87]统计发现模拟一条 ARM 条件转移指令平均需要多达 16 条 RISC-V 指令模拟.为了提高标志位的翻译效率并减少非必要的计算量,QEMU[13]、FX!32[1]、Harmonia[53]等引入了延迟计算技术,将标志位的计算向后拖延到执行阶段.延迟计算有效的减少了代码计算量,但是部分标志位的定值并不会被后续指令使用,此外,标志位状态信息的存储也引入了额外开销.为了进一步降低状态标志位模拟开销,文献[174]提出标志位的模式化翻译方法,翻译时挖掘标志位定值与引用之间的语义关系,选择目标平台上具有相同语义功能的指令组合翻译标志位.文献[87]采用软硬协同的方式,实现源平台与目标平台标志位寄存器的一对一映射,在不修改目标 ISA 和编程模型的前提下实现了标志位的设置和引用操作.Li 等人[87]扩展目标平台 ISA 中算术指令条件位的硬件功能和翻译器对应的 IR 表示,实现源平台与目标平台标志位寄存器的一对一映射.

不失一般性,不同平台对基础函数库的接口设计是一致的.因此,将基础函数库的翻译直接转换为调用目标平台的本地函数库可以避免大量翻译工作.Tan 等人[175]提出在翻译阶段结合可执行文件符号表和链接信息表,将常用库函数翻译转换为本地库函数调用,实验表明对于部分测试课题的性能加速比达 20.9 倍.但其只实现了部分基础库函数的本地化调用.类似地,box86/64[60]采用更加全面的函数库本地化,将系统中常用的库函数全部实现了封装和本地化调用.函数库本地化在大量成熟的翻译系统设计中被广泛使用,如 Instrew[14]和 FEX-emu[38]均采用了函数库本地化设计思想.然而,在函数库本地化时涉及到对被替换函数的符号解析和查询匹配,减弱了库函数本地化的优化效果.Fu 等人[176]提出将函数库查询信息做静态预处理,并使用散列函数优化查询过程,降低了库函数处理的查询开销.

除了上述优化,研究发现在构建二进制翻译系统时,其依赖的基础编译器优化能力也会影响翻译代码生成质量.本文基于 GCC 编译器分别在命令行选项"-O0"和"-O2"下构建 QEMU,然后分别利用构建生成的 QEMU 完成 x86-64-to-ARM64 的二进制翻译,测试集选用 SPEC2006 和 Windows 7 二进制应用.表 8 列出了基于"-O2"选项相比"-O0"选项构建的 QEMU 翻译器的翻译效率加速比,测试结果表明采用更优的编译优化选项(-O2)会显著影响二进制翻译效率.据我们调研所知,目前针对此部分的研究较少.

表 8 QEMU 基于"-O0"和"-O2"选项构建时翻译性能加速比

测试程序	"-O2"/"-O0"选项加速比
SPEC INT2006	1.46
SPEC FP2006	7.18
Windows 7	2.5

4.4 软硬协同优化

采用软件模拟的二进制翻译方法较好屏蔽了硬件差异,保证了源程序语义逻辑的正确翻译.但是软件模拟的指令执行速度要比硬件直接支持慢很多,甚至存在数量级上的差异[17].为了减少硬件之间的差异,有大量研究提出在目标平台硬件上直接支持翻译所需要的部件,实现软硬协同的二进制翻译加速是近年来较为流行的设计方法.根据与处理器的耦合度不同,软硬协同优化分为异构协同加速和处理器协同支持.

4.4.1 异构协同加速

异构协同加速通过引入异构加速部件来弥补处理器翻译或运行时开销大的不足.基于 FPGA、CGRA 和 GPU 等异构设备实现不同 ISA 应用程序二进制翻译协同加速的方法已被业界广泛使用.协处理器与主处理器协同工作,减少了主处理器流水线中的空洞.针对标志寄存器低效模拟问题,Yao 等人[145]基于 FPGA 增加硬件寄存器,加速对 x86 状态标志位的模拟.针对跨平台指令差异导致的代码质量差问题,Chai 等人[177]利用 FPGA 进行灵活的硬件指令动态可重构设计,实现不同类型指令流的硬件协同支持.Ramon 等人[178]提出将二进制翻译过程中的热路径代码映射到 CGRA 开展并行加速.类似地,文献[179]提出 CGRA 和 ARM NEON 混合使用的加速翻译模型,硬件协同支持指令级并行和数据级并行.此外,GPU 强大的计算能力也为二进制翻译加速提供了硬件支撑.Dong 等人[73]基于 GPU 进行软硬协同二进制翻译技术的尝试,提出将无数据依赖的可并行代码派发至 GPU

运行,对于 GPU 上无法运行的代码再在 CPU 上利用二进制翻译执行.

4.4.2 处理器协同支持

异构协同加速的方法类似于一种即插即用的硬件加速,其在灵活性上表现较好,可以根据性能瓶颈点进行针对性协同加速.但是异构加速部件与 CPU 处理器之间的数据通信会产生可观的延迟开销,导致翻译效果大打折扣.为了进一步提升硬件加速效果,有研究提出在目标平台硬件上直接支持二进制翻译所需的翻译部件.针对源平台和目标平台体系结构差异大的问题,LoongArch 在原有龙芯指令系统基础上增加了 MIPS 不具备但 x86 和 ARM 具备的核心功能,在指令功能、运行时环境、核心态功能等方面实现处理器的硬件扩展[17].测试 Linux 操作系统启动时间,结果表明基于软硬协同加速后龙芯二进制翻译器性能提升了 21 倍,翻译效率达到 79.8%[92].为了高效保证内存一致性,Rosetta 2[180]在 ARM 硬件上支持了 x86 的强内存模型,当翻译 x86 代码时会自动切换到强内存模型,而运行 ARM 程序时再切回到其原生内存模型,硬件支持的内存模型有效降低了并发程序翻译时内存模型变换带来的开销.

嵌入式和移动互联网的发展促进了不同类型市场的应用之间数据融合,这种市场融合加速了 RISC 和 CISC 架构之间应用的转换与翻译[58, 59, 146].针对 RISC 向 CISC 架构翻译时面临的状态标志位功能不对等问题,Harmonia[53]采用 MOVBE 硬件指令加速状态标志位计算.文献[24]面向申威平台引入标志位计算、浮点运算、多媒体等指令的硬件协同设计,提升了从 x86 到申威平台指令翻译的代码质量.

此外,有研究利用硬件加速提升程序的并行度.Transmeta[4]利用 Crusoe 处理器的 VLIW 指令特性,重组翻译后的二进制代码以实现同步执行.Crusoe 增加了特殊的硬件功能来检测同一个地址的“读-写-读”冲突,支持对访存指令进行激进式的代码调度.文献[181]引入硬件加速部件实现指令翻译和调度的充分加速.文献[66]在文献[181]的基础上实现 RISC-V-to-VLIW,增加双 VLIW 核心进一步提升翻译效率.文献[182]在文献[66]的基础上增加访存队列和掩码模块,支持在 VLIW 中指令乱序执行.

4.5 小结

本节分别从翻译开销优化、运行时优化、代码生成优化和软硬协同优化等角度分析了二进制翻译常见优化方法.总结发现,翻译开销优化可以有效卸载翻译器自身在翻译和优化时所带来的性能开销,是当前业界普遍采用的优化手段.如 ExaGear[12]、Rosetta 2[5]、Tango[37]等均采用了相关优化;运行时优化借助动态优化技术,对于提升动态翻译效率效果显著.然而,由于静态翻译在运行时信息获取和分支跳转计算等方面存在不足,实施相关优化具有局限性;代码生成优化有效利用目标平台特性来提升翻译后代码生成质量,改进了部分指令低效模拟的不足.源平台和目标平台体系结构上的差异是限制二进制翻译效率提升的根本原因,无论采用何种软件优化手段,基于软件模拟的 CPU 指令执行相比硬件直接执行在翻译依旧存在较大差距.软硬协同的二进制翻译优化作为提升二进制翻译效率的最有效手段,被广泛关注.不足的是,软硬协同的优化方法需要修改处理器硬件,在设计灵活性方面存在较高的门槛.

5 二进制翻译技术应用领域

二进制翻译技术的研究动力归结于其蕴含的潜在商业价值和市场需求.目前,二进制翻译在程序分析与优化、软件迁移、安全研究等众多场景中均发挥着重要作用.本节对二进制翻译技术的具体应用场景进行介绍.

5.1 软件迁移

软件迁移通过相关技术手段将已有的软件从一个平台迁移到另外一个新的平台上.将旧体系结构平台遗留软件直接迁移到新体系结构平台是二进制翻译最主要的应用场景.在历史上,Alpha[1]、Tansmeta[4]、HP[2]、Apple[5]、Intel[39]等均采用此方法协助推出新架构,当前市场上也活跃着一大批实践案例.

微软为了实现向后兼容,在 Windows 64 位操作系统上推出使用 WOW64[183]模拟器.WOW64 模拟器使得大多数 Windows 32 位应用程序无需修改即可在 Windows 64 位版本上运行.为了在 ARM 平台支持 Windows 系统,微软与高通联合推出了 Win10 ARM 完整版,基于二进制翻译技术将 Windows/x86 软件生态直接移植到高通

的 ARM64 平台上运行[184],保证了 Windows/ARM 笔记本用户可以享受到更广泛的软件选择,提升了 ARM 设备的兼容性和可用性.苹果公司利用 Rosetta 2[5]翻译器在 Apple Silicon M1 上高效运行 x86-64 指令集的二进制代码,实现了终端与个人 PC 从 x86-64 到 ARM 架构上的生态融合.华为利用 ExaGear[12]实现 x86(32/64 位)或 ARM32 平台应用到 ARM64 平台的迁移,极大拓展了鲲鹏服务器的软件生态圈.开源 FEX-Emu[38]实现了 Speedy x86/x86-64 游戏在 AArch64 平台运行.龙芯中科推出 LAT[17]系列二进制翻译器实现 x86、ARM、MIPS 等指令系统的二进制应用在 LoongArch 兼容运行,有效弥补了其软件生态的不足.此外,LoongArch 结合 LAT 和 Wine[185],实现在 Linux/LoongArch 上运行 Windows/x86 的应用程序,并完成大批 Linux 老旧打印机等办公设备利旧.

目前,Android/ARM 的强大应用市场催生了大量的 Android 模拟器.Houdini[39]实现了 Android/ARM 应用在英特尔处理器上高效翻译运行.Dolphin[186]支持模拟 GameCube/Wii 游戏在 Android 上运行.DAOW 模拟器[59]实现 Windows PC 上高效运行大型 3D Android 游戏,并有效解决了完全虚拟化带来的性能开销问题.谷歌推出的 Android 11 模拟器[187]支持 ARM 应用在 x86 平台的台式机、笔记本、服务器以及云环境上高效运行.英特尔与腾讯携手推出应用宝,通过 Bridge 技术[188]和 Celadon 技术[189]联合驱动,实现个人 PC 和移动应用之间的体验打通,进一步促进了 Android 与 Windows 的生态融合.

5.2 程序分析与优化

传统静态编译器在编译阶段难以对程序进行全局分析,无法准确预测程序的实际执行行为.此外,对于使用动态加载、共享库和运行时绑定技术的程序来说,编译器静态分析优化的优势较弱.相反,动态二进制翻译技术可以利用代码全局信息开展优化,弥补了传统编译器在此方面的不足.二进制翻译利用 LTO(link time optimization)、PGO(profile-guarded optimizations)等技术实现代码内联、反馈优化、死代码删除等优化操作.Paulino 等人[190]研究发现,对于经过 GCC 和 LLVM 等编译器高度优化后的程序,继续施加二进制分析与优化,依旧可以实现 10%以上的性能加速效果.

近年来,一些研究者致力于利用二进制翻译技术进行程序分析与优化,并产生了大量的研究成果.Luk 等人[138]提出基于采样的优化工具 Ispike,通过将二进制程序运行时的信息反馈给优化器对程序进行优化,降低了指令和数据的访存延迟.Panchenko 等人[191]提出用于数据中心及其他领域的实用二进制优化器 BOLT,其建立在 LLVM 编译器基础架构之上,利用基于采样的运行时信息对二进制文件进行链接后优化,最终实现代码优化布局与重写.谷歌提出的 Propeller 技术[192]对二进制程序对应的 IR 进行分析与优化,并基于运行时信息对程序进行基本块重排、函数划分、函数重排,可以达到与 BOLT 优化器相近的优化效果.Propeller 与 BOLT 的主要区别在于:(1)BOLT 属于链接后优化,Propeller 则是链接时优化.(2)BOLT 的输入是二进制和运行时信息,而 Propeller 的输入是缓存的 IR 中间表示文件和运行时信息.Zou 等人[193]基于 IR 对嵌入式全系统软件进行动态二进制插桩,实现嵌入式全系统软件的运行控制流跟踪.该方法以基本块为单位开展细粒度插桩,扩大了二进制插桩在程序分析领域的应用范围.DynamoRIO[42]基于热路径优化构建代码块,然后对二进制代码引入窥孔优化、常量传播、循环展开等优化,经过优化后部分应用翻译效率优于本地原生执行.Samuel 等人[194]提出矢量化二进制翻译器 VectorVisor,其提供了一个平面内存模型的抽象,在 GPU 上自动交错底层虚拟机的地址空间,为 GPU 加速提供了新的机会.

5.3 二进制符号执行

符号执行作为一种强大的软件分析和错误检测技术,根据处理对象不同可以分为两种:(1)Source-based 符号执行.在源代码可用的前提下,利用编译器将其编译为 IR 后再植入符号执行信息;(2)Binary-only 符号执行.对于源代码不可用的二进制程序,利用二进制翻译将其变换为 IR 后再植入符号执行信息.然而,在实际情况中大多数程序不提供源代码,基于 Binary-only 二进制符号执行被广泛使用.

Yan 等人[195]提出符号执行翻译器 Angr,它基于 VEX IR 对关键符号信息插桩并符号化解释执行,对于不带符号的代码采用具体执行方式.Angr 具有较强的通用性,但执行速度较慢.为了将 Source-based 符号执行系统

KLEE 扩展到第三方依赖库和操作系统内核,Chipounov 等人[196]创建了 S2E 系统.S2E 使用 QEMU 将目标程序从二进制转换为 TCG IR,之后将包含符号信息的 TCG IR 进一步转换为 LLVM 字节码并传递给 KLEE,最后使用 KLEE 符号化解释执行.S2E 相比于 Angr 的设置和运行更复杂,但程序分析范围更加全面.不足的是,S2E 与 Angr 的性能均较差.QSYM[197]作为一个高性能的二进制符号执行工具,利用动态二进制翻译技术将符号执行与本地原生执行紧密集成,实现更细粒度、更快的指令级符号分析与执行.QSYM 不再依赖目标程序的 IR 做符号插桩,而是直接基于指令级进行符号插桩与翻译,相较于基于 IR 的符号执行具有更强的鲁棒性.然而,QSYM 面向 x86 指令集设计,迁移到其他架构难度较大.

Sebastian 等人提出一种 Source-based 的符号执行器 SYMCC[198],它在程序源代码的编译阶段插入符号信息.实验结果表明,SYMCC 的执行性能相比于 QSYM 可以提升两个数量级,但 SYMCC 基于源码编译插桩的设计方法限制了其面向二进制文件的分析使用.受 SYMCC 启发,SymQEMU[30]关注如何实现基于编译的符号执行在二进制文件上工作.SymQEMU 将二进制翻译和符号执行充分融合,基于 QEMU 设计了面向二进制的符号分析工具,通过插桩 TCG IR 中间代码来建立符号约束表达式.与 Anger 和 S2E 相比,SymQEMU 的执行效率更高.与 QSYM 相比,SymQEMU 在保持高效的同时具有平台可扩展性.此外,测试结果表明 SymQEMU 在执行效率和测试覆盖率方面优于 SymCC 技术.表 9 给出了上述基于二进制翻译的符号执行工具的对比情况.

表 9 基于二进制翻译的符号执行工具对比

工具	实现语言	IR 类型	执行速度	支持多平台	基于二进制程序	跨平台执行
Angr[195]	Python	VEX	×	√	√	√
S2E[196]	C/C++	TCG&LLVM	×	√	√	√
QSYM[197]	C++	无	√	×	√	×
SymCC[198]	C++	LLVM	√	√	×	×
SymQEMU[30]	C/C++	TCG	√	√	√	√

5.4 虚拟化

虚拟化技术可以在一个特定的抽象层中,使用下层系统提供的接口为上层系统实现约定接口的功能.虚拟化技术被应用于子系统、组件以及整个计算机系统.二进制翻译本身作为一种虚拟化技术,其在虚拟化应用中被广泛采纳.

VMware 虚拟平台[199]利用动态二进制翻译技术实现了 x86 架构 CPU 的完全虚拟化,允许多个操作系统环境在 x86 的 PC 机上并发运行.Transmeta 公司推出的 Crusoe 微处理器[4]采用软硬件协同设计的二进制翻译技术实现指令集虚拟化.Cota 等人[15]提出了基于动态二进制翻译的跨指令集多线程模拟器 Pico,它可以在保持速度、可移植性和正确性的同时,在多核主机上高效的模拟多核客户机,支持 64 核的全系统虚拟化.QEMU 在虚拟化中被广泛使用,可以提供对多种源平台的虚拟支持.著名的 KVM[200]和 Google Android[187]等模拟器均是基于 QEMU 的二次开发实现.

Wang 等人[19]结合龙芯 GS464E 处理器和 QEMU 二进制翻译融合,实现了软硬件融合的访存虚拟化优化.Huang 等人[91]利用二进制翻译技术提出了一种名为 BTMMU 的跨 ISA 内存虚拟化方法.Spink 等人[9]利用二进制翻译技术实现了从 ARMv8 客户机到 x86-64 主机的异构虚拟化系统 Captive.

5.5 安全研究

随着网络技术的不断发展,计算机系统的安全问题越来越引起人们的关注.其中程序行为监控、数据流分析、代码访问权限设定等成为研究的焦点.二进制翻译被认为是一种特殊的反向编译技术,近年来被广泛应用于软件安全研究.

内核代码安全.内核代码复杂且庞大,其中驱动程序代码是产生漏洞的主要部分.如何有效地分析内核驱动模块代码是一个严峻的挑战.Cota 等人[172]使用动态二进制翻译技术模拟相同指令集架构或跨指令集架构的虚拟机,并且监控用户态及系统态的所有指令和数据,从而对包括内核在内的整个虚拟机代码进行安全分析.

恶意代码识别.Shan 等人[201]提出了恶意代码检测器 BTMD,它基于动静态结合的二进制翻译技术来检测恶意软件并阻止其执行.Michael 等人[202]提出了 Mobile-Sandbox,它是一种将机器学习与二进制翻译相结合的

代码分析器.Mobile-Sandbox 包含三个步骤:(1)静态分析部分用于分析应用程序,以获得应用程序的总体概况.(2)动态分析部分在模拟器中执行应用程序并记录应用程序的每个操作.(3)将静态分析期间收集的信息应用于机器学习技术,以检测恶意应用程序.

污点分析.动态分析平台 PANDA[35]建立在 QEMU 全系统仿真基础之上,具有记录和重放执行的能力,支持单一的动态污点分析.Cao 等人[203]实现了名为 TimePlayer 的原型系统,TimePlayer 利用 PANDA 提供的记录和重放特性来检测未初始化变量的引用,在检测到差异之后,利用符号化污点分析进一步确定未初始化变量的位置.Valgrind[16]利用动态二进制翻译技术将程序变换为 VEX IR,然后基于影子内存对中间代码做插桩分析,实时跟踪程序内存信息,提供包括内存错误捕获、指针未释放、非法地址访问等程序行为分析.

控制流图恢复.Peter 等人[204]基于 SATURN 混淆工具将二进制代码提升至编译器中间语言 LLVM IR,采用迭代控制流图构造算法恢复混淆的二进制程序的控制流图.Kan 等人[205]提出一种恢复原始控制流图的自动化方法,实现了对 Android 原生二进制代码的反混淆分析.

此外,二进制翻译技术也被应用到逆向工程中.例如文献[206]提出一种构建于程序动态二进制分析基础之上的协议模型逆向提取方法,旨在解决如何根据网络应用程序的动态执行过程逆向获取协议消息格式、协议模型等问题.

5.6 小结

本节分别从软件迁移、程序分析与优化、二进制符号执行、虚拟化、安全研究等方面介绍了二进制翻译技术在现实任务中的典型应用.除了上述应用外,也有研究者致力于将二进制翻译技术应用到其他任务中,例如无序微架构替代[4, 207]、异构移动计算[34]、嵌入式应用迁移[49, 54, 135]、体系结构仿真[49, 179, 181]、VLIW 处理器的并行特性利用[170, 181]、处理器能耗效率提升[160, 208]、内存事务转换等[209].

6 未来方向展望

(1)新型高效翻译框架构建

自从最早期的二进制翻译系统推出以来,二进制翻译技术在性能优化、功能开发、应用领域部署等方面不断突破,取得了丰硕的研究成果.虽然不同任务场景对二进制翻译的需求各不相同,但翻译效率和功能正确性始终是衡量二进制翻译框架优劣的关键指标.

目前,业界已存在多种不同框架的二进制翻译系统,但它们大多针对特定领域或特定处理器设计.由于不同处理器平台之间的差异,这些二进制翻译系统在框架设计和优化上或多或少存在定制化,可扩展性较差.此外,现有的二进制翻译系统对于浮点指令和向量指令的翻译效率较低.开源项目 QEMU 作为业界研究最广泛的翻译器,具有较强的平台可扩展性.然而,QEMU 采用纯软件模拟指令的方法,翻译效率较低,与本地原生执行相比相差甚远.近年来,一些研究者致力于将 LLVM 编译框架引入到二进制翻译研究中.LLVM 编译器具有强大的中间表示以及简洁灵活的编程接口,为新型二进制翻译框架的设计提供了研究基础.然而,基于 LLVM 设计的二进制翻译框架虽然能够生成较高质量的目标代码,但是其在翻译和优化阶段引发的性能开销问题不容忽视.因此,设计一套翻译效率高和功能完备的新型二进制翻译框架仍然是该领域值得探索的方向.

(2)多融合优化

二进制翻译是一项涉及处理器体系结构、指令集、编译器、操作系统等多个领域的综合性技术,充分融合不同领域的优化技术来提升二进制翻译效率是当前的研究热点.

首先,静态翻译和动态翻译各有利弊,充分发挥二者优势,实现动静结合的翻译方法可以降低翻译开销.动静结合在缓存代码管理、翻译时机选取等方面的研究仍值得继续探索;其次,通过融合热路径优化、寄存器映射、代码缓存优化、库函数本地化等多种技术,可以显著提升翻译效率.未来可以充分利用目标平台体系结构特点,将多面体优化、AOT 技术等编译优化融合到翻译优化中;再次,基于软件模拟的指令翻译方法在性能上低于硬件直接执行,特别是对于一些计算密集型、对实时性要求较高的程序,翻译效率距离本地原生编译执行还相差较大.采用异构设备协同加速、处理器硬件支持等多种软硬协同优化,可以缩小不同架构之间硬件差异;最

后,一些研究者将机器学习引入到二进制翻译并取得了显著研究成果,未来基于机器学习生成更加高效的翻译规则、提升指令覆盖率以及实现全系统的翻译等仍然是一个开放性的研究课题.

(3)并行翻译支持

多核处理器的出现为应用程序提供了更多的计算资源.同时,随着并行编程技术的发展,向量化场景变得更加常见.要求二进制翻译既能正确翻译应用程序,又能充分挖掘目标平台处理器的并行特性.

首先,并发应用的普及,要求二进制翻译能正确维护并发程序的原有逻辑,保证被翻译程序在目标平台正确运行.尽管已有研究实现了并发程序翻译的逻辑正确性,但在并发程序的高效翻译方面仍然是研究热点.同时,不同处理器硬件对数据访问顺序的一致性保证差异较大,对并发程序翻译时强弱内存模型一致性保证依旧是二进制翻译的研究热点;其次,多线程翻译模式将程序翻译和执行部署在不同线程,隐藏了翻译自身带来的开销.然而,在利用多线程加速翻译的同时也引入了新的问题,例如线程任务划分、翻译预测与响应策略、共享数据访问方式、缓存代码管理等;最后,大量遗留软件是基于标量指令或者位宽较短的向量指令实现.如果按原语义翻译执行,遗留软件无法充分利用新型处理器架构提供的硬件新特性.利用二进制翻译技术提升指令并行性成为研究热点.当前,在可并行化数据识别、旧向量指令高效翻译重组等方面仍然是一个重要研究方向.此外,在指令并行化过程中涉及地址不对齐处理、并行化收益、寄存器分配、数据依赖关系剖析等依旧是研究热点.

(4)新兴领域应用

二进制翻译技术在软件迁移、程序优化与分析、二进制符号执行、虚拟化、程序安全研究等领域发挥着重要作用.总结发现,已有研究主要是从 CISC 到 RISC 架构的翻译研究,例如从 x86 到 ARM、MIPS、RISC-V 平台的翻译,但是从 ARM、MIPS、RISC-V 平台向 x86 架构的翻译研究相对较少.

随着嵌入式、移动设备、手机业务的不断发展,基于 ARM、RISC-V 等 RISC 架构构建的应用在数据中心部署越来越普遍,打破了传统以 x86 为代表的 CISC 垄断地位.将大量相关二进制应用从 RISC 架构迁移到 CISC 架构运行成为一个新的趋势[53, 58, 59, 67, 88, 99, 136, 140, 146].x86 架构具有丰富的指令集,但是通用寄存器较少.从 ARM、RISC-V 架构到 x86 架构的应用翻译时,通常目标平台的寄存器数量少于源平台寄存器数量,寄存器映射问题变得更具挑战性.此外,x86 架构与 ARM、RISC-V 等架构中的指令标志位设置差异较大,如何在目标平台中计算、存储和设置标志位并生成高效的 x86 目标代码值得深入研究.

此外,现有研究大多集中于通用计算领域的应用翻译,对高吞吐量程序和具有实时需求的新兴物联网领域关注较少,同时移动计算和云计算要求二进制翻译能够在统一框架下实现异构软件调度与执行,这一应用方向也值得探索.

7 总结

二进制翻译技术是实现跨指令系统软硬件兼容的重要手段,在处理器的发展过程中扮演着重要角色.发展至今,二进制翻译系统的种类不断丰富、功能不断完善、翻译效率持续提升,已经取得了大量的研究成果.本文围绕二进制翻译技术的研究现状,梳理了已有的二进制翻译技术.分析总结了二进制翻译技术分类、典型二进制翻译系统、指令翻译方法、关键问题研究以及二进制翻译性能优化等,并讨论了二进制翻译的典型应用场景.最后对二进制翻译技术未来潜在的研究方向进行展望.我们旨在对当前二进制翻译技术的代表性研究工作进行总结,希望这一探索可以为该技术的未来研究工作提供借鉴.

References:

1. R. Hookway.DIGITAL FX!32 running 32-Bit x86 applications on Alpha NT.In: Proceedings IEEE COMPCON 97. Digest of Papers. 1997. 37-42. [doi:10.1109/CMPCON.1997.584668].
2. Cindy Zheng, Carol Thompson.PA-RISC to IA-64 transparent execution, no recompilation. Computer, 2000. 33(3). 47-52. [doi:10.1109/2.825695].
3. Leonid Baraz, Tevi Devor, Orna Etzion, Shalom Goldenberg, Alex Skaletsky, Young Wang, Yigel Zemach.IA-32 execution layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems.In: Proceedings. 36th Annual IEEE/ACM

International Symposium on Microarchitecture.MICRO. 2003. 191-201. [doi:10.1109/MICRO.2003.1253195].

4. J. C Dehnert, B. K Grant, J. P Banning, R Johnson, T Kistler, A Klaiber, J Mattson.The transmeta Code Morphing (TM) Software: Using speculation, recovery, and adaptive retranslation to address real-life challenges.In: International Symposium on Code Generation and Optimization. San Francisco, Ca. 2003. 15-24.
5. Apple. Rosetta 2 on a Mac with Apple silicon. 2021.https://support.apple.com/fr-fr/guide/security/secebb113be1/web.
6. K. Ebcioglu, E. R. Altman, Machinery Assoc Comp.DAISY: dynamic compilation for 100% architectural compatibility.In: 24th Annual International Symposium on Computer Architecture. Denver, Co. 1997. 26-37.
7. Erik R. Altman, Michael K. Gschwind, Sumedh W. Sathaye, Stephen V. Kosonocky, Arthur A. Bright, Jason E. Fritts, BOA: The Architecture of a Binary Translation Processor, in IBM Research Report RC. 1999.
8. Li Jianhui, Ma Xiangning, Zhu Chuanqi.Dynamic Binary Translation and Optimization. Journal of Computer Research and Development, 2007. 44(1). 161, (in Chinese with English abstract).
9. Tom Spink, Harry Wagstaff, Björn Franke.A Retargetable System-level DBT Hypervisor. ACM Trans. Comput. Syst., 2020. 36(4). 1-24. [doi:10.1145/3386161].
10. Hu Shiliang, J. E. Smith.Reducing Startup Time in Co-Designed Virtual Machines.In: 33rd International Symposium on Computer Architecture (ISCA'06). 2006. 277-288. [doi:10.1109/ISCA.2006.33].
11. Fu Shengyu, Hong Dingyong, Liu Yuping, Wu Janjan, Hsu Weichung.Efficient and retargetable SIMD translation in a dynamic binary translator. Software: Practice and Experience, 2018. 48(6). 1312-1330. [doi:https://doi.org/10.1002/spe.2573].
12. Lv Yandong. Principle and application of dynamic binary translation technology. 2021.https://ppt.infoq.cn/slide/show?cid=83&pid=3238.
13. Fabrice Bellard.QEMU, a fast and portable dynamic translator.In: 2005 USENIX Annual Technical Conference. Anaheim, CA. 2005. 41-46. [doi:10.5555/1247360.1247401].
14. Alexis Engelke, Dominik Okwieka, Martin Schulz.Efficient LLVM-based dynamic binary translation.In: Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Virtual, USA: Association for Computing Machinery. 2021. 165–171. [doi:10.1145/3453933.3454022].
15. Emilio G. Cota, Paolo Bonzini, Alex Bennée, Luca P. Carloni.Cross-ISA machine emulation for multicores.In: Proceedings of the 2017 International Symposium on Code Generation and Optimization. Austin, USA: IEEE Press. 2017. 210–220.
16. N. Nethercote, J. Seward.Valgrind: A framework for heavyweight dynamic binary instrumentation. Acm Sigplan Notices, 2007. 42(6). 89-100. [doi:10.1145/1273442.1250746].
17. Hu Weiwu, Wang Wenxiang, Wu Ruiyang, Wang Huandong, Zeng Lu, Xu Chenghua, Gao Xiang, Zhang Fuxin. Loongson Instruction Set Architecture Technology. Journal of Computer Research and Development 2023. 60(1). 2-16, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.202220196].
18. Li Nan, Pang Jianmin.Intermediate code optimization method for binary translation based on intermediate representation rule replacement. Journal of National University of Defense Technology, 2021. 43(4). 156-162.
19. Zhenhua Wang, Guojie Jin, Wenxiang Wang.A Dual-TLB Method for MIPS Heterogeneous Virtualization.In: In The International Symposium on Code Generation and Optimization (CGO) 2015 (AMASBT workshop). 2015.
20. Ding Yong Hong, Chun Chen Hsu, Pen Chung Yew, Jan Jan Wu, Wei Chung Hsu, Pangfeng Liu, Chien Min Wang, Yeh Ching Chung.HQEMU: a multi-threaded and retargetable dynamic binary translator on multicores.In: Proceedings of the Tenth International Symposium on Code Generation and Optimization. San Jose, California: Association for Computing Machinery. 2012. 104–113. [doi:10.1145/2259016.2259030].
21. Feng Yue, Jianmin Pang, Xiaosu Han, Jinxian Cui.An Improved Code Cache Management Scheme from I386 to Alpha In Dynamic Binary Translation.In: Proceedings of the 2010 Second International Conference on Computer Modeling and Simulation. IEEE Computer Society. 2010. 321–324. [doi:10.1109/iccms.2010.97].
22. K. Scott, N. Kumar, S. Velusamy, B. Childers, J. W. Davidson, M. L. Soffa.Retargetable and reconfigurable software dynamic translation.In: International Symposium on Code Generation and Optimization, CGO 2003. 2003. 36-47. [doi:10.1109/CGO.2003.1191531].
23. J. D. Hiser, D. Williams, W. Hu, J. W. Davidson, J. Mars, B. R. Childers.Evaluating Indirect Branch Handling Mechanisms in Software Dynamic Translation Systems.In: International Symposium on Code Generation and Optimization (CGO'07). 2007. 61-73. [doi:10.1109/CGO.2007.10].
24. Xiao Nan Liu. Research on Key Technologies of Binary Translation for the Domestic CPU. Doctor of Engineering. PLA Information Engineering University. 2014. (in Chinese with English abstract).
25. Sheng-Yu Fu, Ding-Yong Hong, Jan-Jan Wu, Pangfeng Liu, Wei-Chung Hsu, Ieee.SIMD Code Translation in an Enhanced HQEMU.In:

21st IEEE International Conference on Parallel and Distributed Systems ICPADS. Melbourne, AUSTRALIA. 2015. 507-514. [doi:10.1109/icpads.2015.70].

26. Qiang Shi, Rongcai Zhao.Floating Point Optimization Based on Binary Translation System QEMU.In: Proceedings of the 2016 2nd Workshop on Advanced Research and Technology in Industry Applications. Atlantis Press. 2016. 1338-1343. [doi:10.2991/wartia-16.2016.278].
27. Wenwen Wang.Helper function inlining in dynamic binary translation.In: Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, Republic of Korea: Association for Computing Machinery. 2021. 107–118. [doi:10.1145/3446804.3446851].
28. Zhaoguo Wang, Ran Liu, Yufei Chen, Xi Wu, Haibo Chen, Weihua Zhang, Binyu Zang.COREMU: a scalable and portable parallel full-system emulator.In: Proceedings of the 16th ACM symposium on Principles and practice of parallel programming. San Antonio, TX, USA: Association for Computing Machinery. 2011. 213–222. [doi:10.1145/1941553.1941583].
29. Jiunhung Ding, Pochun Chang, Weichung Hsu, Yehching Chung.PQEMU: A Parallel System Emulator Based on QEMU.In: Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems. IEEE Computer Society. 2011. 276–283. [doi:10.1109/icpads.2011.102].
30. Sebastian Poeplau, Aurélien Francillon.SymQEMU: Compilation-based symbolic execution for binaries.In: Network and Distributed Systems Security (NDSS) Symposium 2021. [doi:https://dx.doi.org/10.14722/ndss.2021.23118].
31. Ziyi Zhao, Zhang Jiang, Ximing Liu, Xiaoli Gong, Wenwen Wang, Pen-Chung Yew.DQEMU: A Scalable Emulator with Retargetable DBT on Distributed Platforms.In: 49th International Conference on Parallel Processing - ICPP. Edmonton, AB, Canada: Association for Computing Machinery. 2020. Article 7. [doi:10.1145/3404397.3404403].
32. Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood.Pin: building customized program analysis tools with dynamic instrumentation.In: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. Chicago, IL, USA: Association for Computing Machinery. 2005. 190–200. [doi:10.1145/1065010.1065034].
33. Junyuan Zeng, Yangchun Fu, Zhiqiang Lin.PEMU: A Pin Highly Compatible Out-of-VM Dynamic Binary Instrumentation Framework.In: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul, Turkey: Association for Computing Machinery. 2015. 147–160. [doi:10.1145/2731186.2731201].
34. Mingliang Li, Jianmin Pang, Feng Yue, Jun Wang, Jie Tan.Enhancing Dynamic Binary Translation in Mobile Computing by Leveraging Polyhedral Optimization. Wireless Communications and Mobile Computing, 2021. 2021. 1-12. [doi:10.1155/2021/6611867].
35. Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, Ryan Whelan.Repeatable Reverse Engineering with PANDA.In: Proceedings of the 5th Program Protection and Reverse Engineering Workshop. Los Angeles, CA, USA: Association for Computing Machinery. 2015. Article 4. [doi:10.1145/2843859.2843867].
36. Haibing Guan, Ruhui Ma, Hongbo Yang, Yindong Yang, Liang Liu, Ying Chen.MTCrossBit: A dynamic binary translation system based on multithreaded optimization. Science China Information Sciences, 2011. 54(10). 2064-2078. [doi:10.1007/s11432-011-4414-5].
37. Amanieu D'antras. The Tango binary translation technology. 2019.https://www.amanieusystems.com/technology.
38. Ryan & Scott. fex-emu. 2022.https://fex-emu.org/.
39. Intel Core Processors and Intel Bridge Technology Unleash Windows 11 Experience. https://www.intel.com/content/www/us/en/newsroom/news/intel-tech-unleashes-windows-experience.html.
40. Rodrigo C. O. Rocha, Dennis Sprokholt, Martin Fink, Redha Gouicem, Tom Spink, Soham Chakraborty, Pramod Bhatotia.Lasagne: a static binary translator for weak memory model architectures.In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. San Diego, CA, USA: Association for Computing Machinery. 2022. 888–902. [doi:10.1145/3519939.3523719].
41. Xiangning Ma. Research on Design and Optimization of Binary Translation System. Doctor of Engineering. 2004. (in Chinese with English abstract).
42. Derek L. Bruening. Efficient, Transparent, and Comprehensive Runtime Code Manipulation. PhD. Massachusetts Institute of Technology. 2004.
43. Bob Cmelik, David Keppel.Shade: a fast instruction-set simulator for execution profiling.In: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems. Nashville, Tennessee, USA: Association for Computing Machinery. 1994. 128–137. [doi:10.1145/183018.183032].
44. A. Chernoff, M. Herdeg, R. Hookway, C. Reeve, N. Rubin, T. Tye, S. Bharadwaj Yadavalli, J. Yates.FX!32 a profile-directed binary translator. IEEE Micro, 1998. 18(2). 56-64. [doi:10.1109/40.671403].

45. Amitabh Srivastava, Andrew Edwards, Hoi Vo.Vulcan Binary transformation in a distributed environment. 2001.

46. B. Scholz, M. Probst, A. Krall.Register Liveness Analysis for Optimizing Dynamic Binary Translation.In: Ninth Working Conference on Reverse Engineering, 2002. Proceedings. 2002. 35–44. [doi:10.1109/wcre.2002.1173062].

47. Cristina Garcia Cifuentes, Brian T. Lewis, David Ung, Walkabout: a retargetable dynamic binary translation framework, Sun Microsystems Inc, Editor. 2002: CA, USA.

48. Cheng Wang, Shiliang Hu, Ho-Seop Kim, Sreekumar R. Nair, Mauricio Breternitz, Zhiwei Ying, Youfeng Wu.StarDBT: An Efficient Multi-platform Dynamic Binary Translation System.In: Advances in Computer Systems Architecture. Berlin, Heidelberg: Springer Berlin Heidelberg. 2007. 4-15.

49. Daniel Jones, Nigel Topham.High Speed CPU Simulation Using LTU Dynamic Binary Translation.In: High Performance Embedded Architectures and Compilers. Berlin, Heidelberg: Springer Berlin Heidelberg. 2009. 50-64.

50. Ryan W. Moore, José A. Baiocchi, Bruce R. Childers, Jack W. Davidson, Jason D. Hiser.Addressing the challenges of DBT for the ARM architecture. SIGPLAN Not., 2009. 44(7). 147–156. [doi:10.1145/1543136.1542472].

51. Maxwell Souza, Daniel Nicácio, Guido Araújo.ISAMAP: Instruction Mapping Driven by Dynamic Binary Translation.In: Computer Architecture, ISCA'10. Berlin, Heidelberg: Springer Berlin Heidelberg. 2012. 117-138. [doi:10.1007/978-3-642-24322-6_11].

52. Igor Böhm, Tobias J.K. Edler Von Koch, Stephen C. Kyle, Björn Franke, Nigel Topham.Generalized just-in-time trace compilation using a parallel task farm in a dynamic binary translator.In: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. San Jose, California, USA: Association for Computing Machinery. 2011. 74–85. [doi:10.1145/1993498.1993508].

53. Guilherme Ottoni, Thomas Hartin, Christopher Weaver, Jason Brandt, Belliappa Kuttanna, Hong Wang.Harmonia: a transparent, efficient, and harmonious dynamic binary translator targeting the Intel® architecture.In: Proceedings of the 8th ACM International Conference on Computing Frontiers. Ischia, Italy: Association for Computing Machinery. 2011. 1-10. [doi:10.1145/2016604.2016635].

54. Bor-Yeh Shen, Jiunn-Yeu Chen, Wei-Chung Hsu, Wuu Yang, Acm.LLBT: An LLVM-based Static Binary Translator.In: ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES). Tampere, FINLAND. 2012. 51-60.

55. Haibing Guan, Erzhou Zhu, Hongxi Wang, Ruhui Ma, Yindong Yang, Bin Wang.SINOF: A dynamic-static combined framework for dynamic binary translation. Journal of Systems Architecture, 2012. 58(8). 305-317. [doi:10.1016/j.sysarc.2012.05.002].

56. Xiaochun Zhang, Qi Guo, Yunji Chen, Tianshi Chen, Weiwu Hu, Ieee.HERMES: A Fast Cross-ISA Binary Translator with Post-Optimization.In: Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO). San Francisco, CA. 2015. 246-256.

57. Amanieu D'antras, Cosmin Gorgovan, Jim Garside, Mikel Luján.Low overhead dynamic binary translation on ARM.In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. Barcelona, Spain: Association for Computing Machinery. 2017. 333–346. [doi:10.1145/3062341.3062371].

58. M. Clark and B. Hoult.rv8 : a high performance RISC-V to x 86 binary translator.In: First Workshop on Computer Architecture Research with RISC-V 2017. [doi:10.13140/RG.2.2.30957.69601].

59. Qifan Yang, Zhenhua Li, Yunhao Liu, Hai Long, Yuanchao Huang, Jiaming He, Tianyin Xu, Ennan Zhai.Mobile Gaming on Personal Computers with Direct Android Emulation.In: The 25th Annual International Conference on Mobile Computing and Networking. Los Cabos, Mexico: Association for Computing Machinery. 2019. Article 19. [doi:10.1145/3300061.3300122].

60. Ptitseb. Box64. 2021.https://github.com/ptitSeb/box64.

61. Redha Gouicem, Dennis Sprokholt, Jasper Ruehl, Rodrigo C. O. Rocha, Tom Spink, Soham Chakraborty, Pramod Bhatotia.Risotto: A Dynamic Binary Translator for Weak Memory Model Architectures.In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1. Vancouver, BC, Canada: Association for Computing Machinery. 2022. 107–122. [doi:10.1145/3567955.3567962].

62. Emmett Witchel, Mendel Rosenblum.Embra: fast and flexible machine simulation. SIGMETRICS Perform. Eval. Rev., 1996. 24(1). 68–79. [doi:10.1145/233008.233025].

63. Matthew Chapman, Daniel J. Magenheimer, Parthasarathy Ranganathan, MagiXen: Combining Binary Translation and Virtualization. 2007: Enterprise Systems and Software Laboratory, HP Laboratories Palo Alto.

64. Prashanth P. Bungale, Chi-Keung Luk.PinOS: a programmable framework for whole-system dynamic instrumentation.In: Proceedings of the 3rd international conference on Virtual execution environments. San Diego, California, USA: Association for Computing Machinery. 2007. 137–147. [doi:10.1145/1254810.1254830].

65. R. A. Sokolov, A. V. Ermolovich.Background optimization in full system binary translation. Programming and Computer Software, 2012. 38(3). 119-126. [doi:10.1134/S0361768812030073].

66. S. Rokicki, E. Rohou, S. Derrien.Hybrid-DBT: Hardware/Software Dynamic Binary Translation Targeting VLIW. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019. 38(10). 1872-1885. [doi:10.1109/TCAD.2018.2864288].

67. S. Riedel, F. Schuiki, P. Scheffler, F. Zaruba, L. Benini.Banshee: A Fast LLVM-Based RISC-V Binary Translator.In: 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). 2021. 1-9. [doi:10.1109/ICCAD51958.2021.9643546].

68. S. Y. Fu, J. J. Wu, W. C. Hsu.Improving SIMD code generation in QEMU.In: Automation & Test in Europe Conference & Exhibition (DATE). 2015. 1233-1236. [doi:10.5555/2755753.2757098].

69. Thomas Dullien, Sebastian Porst.REIL: A platform-independent intermediate representation of disassembled code for static code analysis.In: Computer Science. 2009.

70. Hex-Rays Sa. IDA-Pro: State-of-the-art binary code analysis tools. 2017.https://www.hex-rays.com/ida-pro/.

71. David Brumley, Ivan Jager, Thanassis Avgerinos, Edward J. Schwartz.BAP: A Binary Analysis Platform.In. Berlin, Heidelberg: Springer Berlin Heidelberg. 2011. 463-469.

72. Niranjan Hasabnis, R. Sekar.Lifting Assembly to Intermediate Representation: A Novel Approach Leveraging Compilers. SIGARCH Comput. Archit. News, 2016. 44(2). 311–324. [doi:10.1145/2980024.2872380].

73. G. Dong, K. Chen, E. Zhu, Y. Zhang, Z. Qi, H. Guan.A Translation Framework for Virtual Execution Environment on CPU/GPU Architecture.In: 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming. 2010. 130-137. [doi:10.1109/PAAP.2010.53].

74. C. Lattner, V. Adve.LLVM: a compilation framework for lifelong program analysis & transformation.In: International Symposium on Code Generation and Optimization, CGO. 2004. 75-86. [doi:10.1109/CGO.2004.1281665].

75. Alexis Engelke, Martin Schulz.Instrew: leveraging LLVM for high performance dynamic binary instrumentation.In: Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Lausanne, Switzerland: Association for Computing Machinery. 2020. 172–184. [doi:10.1145/3381052.3381319].

76. Lifting Bits. Framework for lifting x86, amd64, and aarch64 program binaries to LLVM bitcode. 2022. https://github.com/liftingbits/mcsema.

77. S. Bharadwaj Yadavalli, Aaron Smith.Raising binaries to LLVM IR with MCTOLL (WIP paper).In: Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems. Phoenix, AZ, USA: Association for Computing Machinery. 2019. 213–218. [doi:10.1145/3316482.3326354].

78. Vitaly Chipounov, George Candea.Dynamically Translating x86 to LLVM using QEMU.In: Ecole Polytechnique Fédérale de Lausanne. Switzerland. 2010.

79. Yi-Ping You, Tsung-Chun Lin, Wuu Yang.Translating AArch64 Floating-Point Instruction Set to the x86-64 Platform.In: Workshop Proceedings of the 48th International Conference on Parallel Processing. Kyoto, Japan: Association for Computing Machinery. 2019. Article 12. [doi:10.1145/3339186.3339192].

80. Nicholas Nethercote. Dynamic Binary Analysis and Instrumentation. Doctor of Philosophy. University of Cambridge, Trinity College. 2004.

81. Wenwen Wang, Stephen Mccamant, Antonia Zhai, Pen-Chung Yew.Enhancing Cross-ISA DBT Through Automatically Learned Translation Rules. SIGPLAN Not., 2018. 53(2). 84–97. [doi:10.1145/3296957.3177160].

82. Changheng Song, Wenwen Wang, Penchung Yew, Antonia Zhai, Weihua Zhang.Unleashing the power of learning: an enhanced learning-based approach for dynamic binary translation.In: Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference. Renton, WA, USA: USENIX Association. 2019. 77–89. [doi:10.5555/3358807.3358815].

83. Sorav Bansal, Alex Aiken.Binary translation using peephole superoptimizers.In: Proceedings of the 8th USENIX conference on Operating systems design and implementation. San Diego, California: USENIX Association. 2008. 177–192.

84. Sorav Bansal, Alex Aiken.Automatic generation of peephole superoptimizers. SIGOPS Oper. Syst. Rev., 2006. 40(5). 394–403. [doi:10.1145/1168917.1168906].

85. Jiang Jinhu, Dong Rongchao, Zhou Zhong, Song Changheng, Wang Wenwen, Zhang Weihua.More with Less – Deriving More Translation Rules with Less Training Data for DBTs Using Parameterization.In: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 2020. 415-426. [doi:10.1109/MICRO50266.2020.00043].

86. Haoran Xu, Fredrik Kjolstad.Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode. Proc. ACM Program. Lang., 2021. 5(OOPSLA). Article 136. [doi:10.1145/3485513].

87. Chunqiang Li, Zhiwei Liu, Yunhai Shang, Lenian He, Xiaolang Yan.A Hardware Non-Invasive Mapping Method for Condition Bits in Binary Translation. Electronics, 2023. 12(14). 3014. [doi:10.3390/electronics12143014].

88. Sheng-Yu Fu, Ding-Yong Hong, Yu-Ping Liu, Jan-Jan Wu, Wei-Chung Hsu.Optimizing data permutations in structured loads/stores

translation and SIMD register mapping for a cross-ISA dynamic binary translator. J. Syst. Archit., 2019. 98(C). 173–190. [doi:10.1016/j.sysarc.2019.07.008].

89. Zhe Wang, Jianjun Li, Chenggang Wu, Dongyan Yang, Zhenjiang Wang, Wei-Chung Hsu, Bin Li, Yong Guan.HSPT: Practical Implementation and Efficient Management of Embedded Shadow Page Tables for Cross-ISA System Virtual Machines.In: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul, Turkey: Association for Computing Machinery. 2015. 53–64. [doi:10.1145/2731186.2731188].
90. Antoine Faravelon, Olivier Gruber, Frédéric Pétrot.Optimizing Memory Access Performance Using Hardware Assisted Virtualization in Retargetable Dynamic Binary Translation.In: Euromicro Conference on Digital System Design (DSD). 2017. 40-46. [doi:10.1109/DSD.2017.41].
91. Kele Huang, Fuxin Zhang, Cun Li, Gen Niu, Junrong Wu, Tianyi Liu.BTMMU: an efficient and versatile cross-ISA memory virtualization.In: Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Virtual, USA: Association for Computing Machinery. 2021. 71–83. [doi:10.1145/3453933.3454015].
92. Weiwu Hu, Guojie Jin, Wenxiang Wang, Xiaochun Zhang, Huandong Wang.LoongISA for compatibility with mainstream instruction set architecture. SCIENTIA SINICA Informationis, 2015. 45(4). 459-479. [doi:10.1360/N112014-00300].
93. M. Kristien, T. Spink, B. Campbell, S. Sarkar, I. Stark, B. Franke, I. Böhm, N. Topham.Fast and Correct Load-Link/Store-Conditional Instruction Handling in DBT Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020. 39(11). 3544-3554. [doi:10.1109/TCAD.2020.3013048].
94. Oscar Almer, Igor Böhm, Tobias Edler Von Koch, Björn Franke, Stephen Kyle, Volker Seeker, Christopher Thompson, Nigel Topham.Scalable multi-core simulation using parallel dynamic binary translation.In: 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. 2011. 190-199. [doi:10.1109/SAMOS.2011.6045461].
95. Ragavendra Natarajan, Antonia Zhai.Leveraging Transactional Execution for Memory Consistency Model Emulation. ACM Trans. Archit. Code Optim., 2015. 12(3). Article 29. [doi:10.1145/2786980].
96. Damian Dechev, Peter Pirkelbauer, Bjarne Stroustrup.Understanding and Effectively Preventing the ABA Problem in Descriptor-Based Lock-Free Designs.In: 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. 2010. 185-192. [doi:10.1109/ISORC.2010.10].
97. Alvise Rigo, Alexander Spyridakis, Daniel Raho.Atomic Instruction Translation Towards A Multi-Threaded QEMU.In: ECMS. 2016.
98. Jiang Xiaowu, Chen Xianglan, Wang Huang, Chen Huaping.A Parallel Full-System Emulator for Risc Architure Host.In: Advances in Computer Science and its Applications. Berlin, Heidelberg: Springer Berlin Heidelberg. 2014. 1045-1052. [doi:10.1007/978-3-642-41674-3_145].
99. Ziyi Zhao, Zhang Jiang, Ying Chen, Xiaoli Gong, Wenwen Wang, Penchung Yew.Enhancing Atomic Instruction Emulation for Cross-ISA Dynamic Binary Translation.In: 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2021. 351-362. [doi:10.1109/CGO51591.2021.9370312].
100. Gen Niu, Fuxin Zhang, Xinyu Li.Eliminate the overhead of interrupt checking in full-system dynamic binary translator.In: Proceedings of the 15th ACM International Conference on Systems and Storage. Haifa, Israel: Association for Computing Machinery. 2022. 1–12. [doi:10.1145/3534056.3534939].
101. Soham Chakraborty.Robustness between Weak Memory Models.In: Formal Methods in Computer Aided Design (FMCAD). New Haven, CT, USA. 2021. 173-182. [doi:10.34727/2021/isbn.978-3-85448-046-4_26].
102. Jade Alglave, Luc Maranget, Susmit Sarkar, Peter Sewell.Fences in Weak Memory Models.In: Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg. 2010. 170–205. [doi:10.1007/978-3-642-14295-6_25].
103. D. Lustig, C. Trippel, M. Pellauer, M. Martonosi.ArMOR: Defending against memory consistency model mismatches in heterogeneous architectures.In: 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). 2015. 388-400. [doi:10.1145/2749469.2750378].
104. Byron Hawkins, Brian Demsk, Derek Bruening, Qin Zhao.Optimizing Binary Translation of Dynamically Generated Code.In: Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO). San Francisco, CA. 2015. 68-78. [doi:10.1109/CGO.2015.7054188].
105. Joy Kamunyori.Handling self-modifying code using software dynamic translation.In: Proceedings of the 2007 conference on Diversity in computing. Orlando, Florida: Association for Computing Machinery. 2007. 32. [doi:10.1145/1347787.1347807].
106. Anzhan Liu, Wenqi Wang.ASCMS: An Accurate Self-Modifying Code Cache Management Strategy in Binary Translation.In: 2013 International Conference on Information Science and Cloud Computing Companion. 2013. 405-410. [doi:10.1109/ISCC-C.2013.52].
107. Wenwen Wang, Jiacheng Wu, Xiaoli Gong, Tao Li, Pen-Chung Yew.Improving Dynamically-Generated Code Performance on Dynamic

Binary Translators. SIGPLAN Not., 2018. 53(3). 17–30. [doi:10.1145/3296975.3186413].

108. C. Cifuentes, M. Van Emmerik.Recovery of jump table case statements from binary code.In: Proceedings Seventh International Workshop on Program Comprehension. 1999. 192-199. [doi:10.1109/WPC.1999.777758].

109. Jiunn-Yeu Chen, Wuu Yang, Wei-Chung Hsu, Bor-Yeh Shen, Quan-Huei Ou.On Static Binary Translation of ARM/Thumb Mixed ISA Binaries. ACM Transactions on Embedded Computing Systems, 2017. 16. 1-25. [doi:10.1145/2996458].

110. Marie Badaroux, Frédéric Pétrot.Arbitrary and Variable Precision Floating-Point Arithmetic Support in Dynamic Binary Translation.In: Proceedings of the 26th Asia and South Pacific Design Automation Conference. Tokyo, Japan: Association for Computing Machinery. 2021. 325–330. [doi:10.1145/3394885.3431416].

111. Yesheng Zhi, Yuanyuan Zhang, Juanru Li, Dawu Gu.Security Testing of Software on Embedded Devices Using x86 Platform.In: Collaborate Computing: Networking, Applications and Worksharing. Cham: Springer International Publishing. 2017. 497-504.

112. Jin Wu, Jian Dong, Ruili Fang, Wen Zhang, Wenwen Wang, Decheng Zuo.FADATest: fast and adaptive performance regression testing of dynamic binary translation systems.In: Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, Pennsylvania: Association for Computing Machinery. 2022. 896–908. [doi:10.1145/3510003.3510169].

113. Hui Guo, Zhenjiang Wang, Chenggang Wu, Ruining He.EATBit: Effective automated test for binary translation with high code coverage.In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2014. 1-6. [doi:10.7873/DATE.2014.097].

114. Jin Wu, Jian Dong, Ruili Fang, Wenwen Wang, Decheng Zuo.PerfDBT: Efficient Performance Regression Testing of Dynamic Binary Translation.In: 2020 IEEE 38th International Conference on Computer Design (ICCD). 2020. 389-392. [doi:10.1109/ICCD50377.2020.00071].

115. Harry Wagstaff, Bruno Bodin, Tom Spink, Björn Franke.SimBench: A portable benchmarking methodology for full-system simulators.In: 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 2017. 217-226. [doi:10.1109/ISPASS.2017.7975293].

116. Soomin Kim, Markus Faerevaag, Minkyu Jung, Seungll Jung, Dongyeop Oh, Jonghyup Lee, Sang Kil Cha.Testing intermediate representations for binary analysis.In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). 2017. 353-364. [doi:10.1109/ASE.2017.8115648].

117. Jiunn Yeu Chen, Wuu Yang, Bor Yeh Shen, Yuan Jia Li, Wei Chung Hsu.Automatic validation for binary translation. Computer Languages, Systems and Structures, 2015. 43(C). 96–115. [doi:10.1016/j.cl.2015.05.002].

118. D. S. Koltunov, V. Yu Efimov, V. A. Padaryan.Automated Testing of a TCG Frontend for Qemu. Programming and Computer Software, 2020. 46(8). 737-746. [doi:10.1134/S0361768820080058].

119. Sandeep Dasgupta, Sushant Dinesh, Deepan Venkatesh, Vikram S. Adve, Christopher W. Fletcher.Scalable validation of binary lifters.In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. London, UK: Association for Computing Machinery. 2020. 655–671. [doi:10.1145/3385412.3385964].

120. Fu Liguo, Pang Jianmin, Wang Jun, Zhang Jiahao, Yue Feng.Formal Model of Correctness and Optimization on Binary Translation. Journal of Computer Research and Development, 2019. 56(9). 2001, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.2019.20180513].

121. Yu-Yuan Chen, Youfeng Wu, Shiliang Hu, Ruby B. Lee.Impact of Dynamic Binary Translators on Security.In: 1st Workshop on Architectural and Microarchitectural Support for Binary Translation, International Symposium on Computer Architecture (ISCA). 2008.

122. Vendicator. StackShield: A "stack smashing" technique protection tool for Linux. 2000.https://www.angelfire.com/sk/stackshield/.

123. Crispin Cowan, Calton Pu, Dave Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grie, Perry Wagle, Qian Zhang.StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks.In: Proceedings of the 7th conference on USENIX Security Symposium 1998. 5. [doi:10.5555/1267549.1267554].

124. Arash Baratloo, Navjot Singh, Timothy Tsai.Transparent run-time defense against stack smashing attacks.In: Proceedings of the annual conference on USENIX Annual Technical Conference. San Diego, California: USENIX Association. 2000. 21. [doi:10.5555/1267724.1267745].

125. Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, Peng Ning.Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software.In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06). 2006. 339-348. [doi:10.1109/ACSAC.2006.9].

126. Bill Horne, Lesley Matheson, Casey Sheehan, Robert E. Tarjan.Dynamic Self-Checking Techniques for Improved Tamper Resistance.In: Security and Privacy in Digital Rights Management. Berlin, Heidelberg: Springer Berlin Heidelberg. 2002. 141-159.

127. Rakan El-Khalil, Angelos D. Keromytis.Hydan: Hiding Information in Program Binaries.In: Information and Communications Security. Berlin, Heidelberg: Springer Berlin Heidelberg. 2004. 187-199.

128. Shankara Pailoor, Xinyu Wang, Hovav Shacham, Isil Dillig.Automated policy synthesis for system call sandboxing. Proc. ACM Program. Lang., 2020. 4(OOPSLA). Article 135. [doi:10.1145/3428203].

129. M. Rajagopalan, M. A. Hiltunen, T. Jim, R. D. Schlichting.System Call Monitoring Using Authenticated System Calls. IEEE Transactions on Dependable and Secure Computing, 2006. 3(3). 216-229. [doi:10.1109/TDSC.2006.41].

130. Hui Xu, Yangfan Zhou, Jiang Ming, Michael Lyu.Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. Cybersecurity, 2020. 3(1). 9. [doi:10.1186/s42400-020-00049-3].

131. Abhijit Mohanta, Anoop Saldanha,Armoring and Evasion: The Anti-Techniques, Mohanta AbhijitSaldanha Anoop. 2020, Apress: Berkeley, CA. 691-720.

132. Roberto Guanciale.Protecting Instruction Set Randomization from Code Reuse Attacks.In: Secure IT Systems. Cham: Springer International Publishing. 2018. 421-436.

133. George Necula,Proof-Carrying Code, van Tilborg Henk C. A.Jajodia Sushil. 2011, Springer US: Boston, MA. 984-986.

134. Wei Chen, Dan Chen, Zhiying Wang.An approach to minimizing the interpretation overhead in Dynamic Binary Translation. The Journal of Supercomputing, 2012. 61(3). 804-825. [doi:10.1007/s11227-011-0636-y].

135. B. Shen, J. You, W. Yang, W. Hsu.An LLVM-based hybrid binary translation system.In: 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12). 2012. 229-236. [doi:10.1109/SIES.2012.6356589].

136. Jin Wu, Jian Dong, Ruili Fang, Ziyi Zhao, Xiaoli Gong, Wenwen Wang, Decheng Zuo.Effective exploitation of SIMD resources in cross-ISA virtualization.In: Proceedings of the 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Virtual, USA: Association for Computing Machinery. 2021. 84–97. [doi:10.1145/3453933.3454016].

137. Wu Youfeng, J. Gregory Steffan, Cristiana Amza.Lengthening Traces to Improve Opportunities for Dynamic Optimization.In: Computer Science. 2007. [doi:10.1.1.136.7737].

138. C. K. Luk, R. Muth, Patil Harish, R. Cohn, G. Lowney.Ispike: a post-link optimizer for the Intel Itanium architecture.In: International Symposium on Code Generation and Optimization, 2004. CGO 2004. 2004. 15-26. [doi:10.1109/CGO.2004.1281660].

139. Amanieu D'antras, Cosmin Gorgovan, Jim Garside, Mikel Luján.Optimizing Indirect Branches in Dynamic Binary Translators. Acm Transactions on Architecture and Code Optimization, 2016. 13(1). 1-25. [doi:10.1145/2866573].

140. Y. P. Liu, D. Y. Hong, J. J. Wu, S. Y. Fu, W. C. Hsu, Ieee.Exploiting Asymmetric SIMD Register Configurations in ARM-to-x86 Dynamic Binary Translation.In: 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). Portland, OR. 2017. 343-355. [doi:10.1109/pact.2017.15].

141. D. Y. Hong, S. Y. Fu, Y. P. Liu, J. J. Wu, W. C. Hsu.Exploiting Longer SIMD Lanes in Dynamic Binary Translation.In: 22nd IEEE International Conference on Parallel and Distributed Systems (ICPADS). Wuhan, PEOPLES R CHINA. 2016. 853-860. [doi:10.1109/icpads.2016.113].

142. Ding Yong Hong, Yu Ping Liu, Sheng Yu Fu, Jan Jan Wu, Wei Chung Hsu.Improving SIMD Parallelism via Dynamic Binary Translation. ACM Transactions on Embedded Computing Systems, 2018. 17(3). 1-27. [doi:10.1145/3173456].

143. Liang Yi, Yuanhua Shao, Guowu Yang, Jinzhao Wu.Register Allocation for QEMU Dynamic Binary Translation Systems. International Journal of Hybrid Information Technology, 2015. 8. 199-210. [doi:10.14257/ijhit.2015.8.2.18].

144. Wen Yanhua, Tang Daguo, Qi Fengbin.Register Mapping and Register Function Cutting out Implementation in Binary Translation. Journal of Software, 2009. 20(zk). 1-7, (in Chinese with English abstract).

145. Y. Yao, Z. Lu, Q. Shi, W. Chen.FPGA based hardware-software co-designed dynamic binary translation system.In: 2013 23rd International Conference on Field programmable Logic and Applications. 2013. 1-4. [doi:10.1109/FPL.2013.6645554].

146. Jyun-Kai Lai, Wuu Yang.Hyperchaining Optimizations for an LLVM-Based Binary Translator on x86-64 and RISC-V Platforms.In: ICPP Workshops '21: 50th International Conference on Parallel Processing Workshop. Association for Computing Machinery. 2021. 1-9. [doi:10.1145/3458744.3473348].

147. Sun Lianshan, Wu Yanjin, Li Linxiangyi, Zhang Changbin, Tang Jingyan.A Dynamic and Static Binary Translation Method Based on Branch Prediction. Electronics, 2023. 12(14). 3025. [doi:10.3390/electronics12143025].

148. Jun Wang, Jianmin Pang, Xiaonan Liu, Feng Yue, Jie Tan, Liguo Fu.Dynamic Translation Optimization Method Based on Static Pre-Translation. IEEE Access, 2019. 7. 21491-21501. [doi:10.1109/ACCESS.2019.2897611].

149. Ma Ruhui, Guan Haibing, Zhu Erzhou, Yang Hongbo, Yang Yindong, Liang Alei.Partitioning the Conventional DBT System for Multiprocessors. Journal of Computer Science and Technology, 2011. 26(3). 474-490. [doi:10.1007/s11390-011-1148-1].

150. Jikun Liu, Gaojin Cao, Hongguang Zhang.Research on pipeline-based dynamic binary translation. Proceedings of 2013 3rd International Conference on Computer Science and Network Technology, 2013. 601-604. [doi:10.1109/ICCSNT.2013.6967185].

151. X. Tu, H. Jin, Z. Yu, J. Chen, Y. Hu.MT-BTRIMER: A Master-Slave Multi-threaded Dynamic Binary Translator.In: 2010 Fifth

International Conference on Frontier of Computer Science and Technology. 2010. 51-56. [doi:10.1109/FCST.2010.72].

152. Bob Wescott, The Every Computer Performance Book, Chapter 3: Useful laws. 2013: CreateSpace Independent Publishing Platform.

153. Ding-Yong Hong, Jan-Jan Wu, Yu-Ping Liu, Sheng-Yu Fu, Wei-Chung Hsu.Processor-Tracing Guided Region Formation in Dynamic Binary Translation. ACM Trans. Archit. Code Optim., 2018. 15(4). Article 52. [doi:10.1145/3281664].

154. Jyun-Siang Huang, Wuu Yang, Yi-Ping You.Profile-guided optimisation for indirect branches in a binary translator. Connection Science, 2022. 34(1). 749-765. [doi:10.1080/09540091.2022.2041555].

155. Ding-Yong Hong, Jan-Jan Wu, Pen-Chung Yew, Wei-Chung Hsu, Chun-Chen Hsu, Pangfeng Liu, Chien-Min Wang, Yeh-Ching Chung.Efficient and Retargetable Dynamic Binary Translation on Multicores. IEEE Trans. Parallel Distrib. Syst., 2014. 25(3). 622–632. [doi:10.1109/tpds.2013.56].

156. Wei Chen, Zhiying Wang, Qiang Dou, Yongwen Wang.A Novel Chaining Approach to Indirect Control Transfer Instructions.In: Availability, Reliability and Security for Business, Enterprise and Health Information Systems. Berlin, Heidelberg: Springer Berlin Heidelberg. 2011. 309-320.

157. Johannes Kinder, Florian Zuleger, Helmut Veith.An Abstract Interpretation-Based Framework for Control Flow Reconstruction from Binaries.In: VMCAI. 2009.

158. Wang Jun, Pang Jianmin, Fu Liguo, Yue Feng, Zhang Jiahao.An Efficient Feedback Static Binary Translator for Solving Indirect Branch. Journal of Computer Research and Development, 2019. 56(4). 742, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.2019.20170412].

159. A. Di Federico, G. Agosta.A jump-target identification method for multi-architecture static binary translation.In: 2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES). 2016. 1-10. [doi:10.1145/2968455.2968514].

160. Wenwen Wang, Pen-Chung Yew, Antonia Zhai, Stephen Mccamant, Youfeng Wu, Jayaram Bobba.Enabling Cross-ISA Offloading for COTS Binaries.In: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services. Niagara Falls, New York, USA: Association for Computing Machinery. 2017. 319–331. [doi:10.1145/3081333.3081337].

161. Wenwen Wang, Pen Chung Yew, Antonia Zhai, Stephen Mccamant.A general persistent code caching framework for dynamic binary translation (DBT).In: Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference. Denver, CO, USA: USENIX Association. 2016. 591–603. [doi:10.5555/3026959.3027013].

162. A. H. Ibrahim, M. B. Abdelhalim, H. Hussein, A. Fahmy.Analysis of x86 instruction set usage for Windows 7 applications.In: 2010 2nd International Conference on Computer Technology and Development. 2010. 511-516. [doi:10.1109/ICCTD.2010.5645851].

163. Wang Jun, Pang Jianmin, Fu Liguo, Yue Feng, Shan Zheng, Zhang Jiahao.A Dynamic and Static Combined Register Mapping Method in Binary Translation. Journal of Computer Research and Development, 2019. 56(4). 708-718. [doi:10.7544/issn1000-1239.2019.20170905].

164. Jun Wang, Jianmin Pang, Liguo Fu, Zheng Shan, Feng Yue, Jiahao Zhang.A Binary Translation Backend Registers Allocation Algorithm Based on Priority.In: Geo-Spatial Knowledge and Intelligence. Singapore: Springer Singapore. 2018. 414-425. [doi:10.1007/978-981-13-0896-3_41].

165. Antoine Faravelon, Olivier Gruber, Frédéric Pétrot,Removing Load/Store Helpers in Dynamic Binary Translation. 2021. 133-160.

166. Jin Wu, Jian Dong, Ruili Fang, Wen Zhang, Wenwen Wang, Decheng Zuo.WDBT: Wear Characterization, Reduction, and Leveling of DBT Systems for Non-Volatile Memory.In: Proceedings of the International Symposium on Memory Systems. Washington DC, DC, USA: Association for Computing Machinery. 2022. 1-13. [doi:10.1145/3488423.3519337].

167. N. Hallou, E. Rohou, P. Clauss, A. Ketterlin.Dynamic re-vectorization of binary code.In: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). 2015. 228-237. [doi:10.1109/SAMOS.2015.7363680].

168. T. Nakamura, S. Miki, S. Oikawa.Automatic Vectorization by Runtime Binary Translation.In: 2011 Second International Conference on Networking and Computing. 2011. 87-94. [doi:10.1109/ICNC.2011.21].

169. C. M. Lin, S. Y. Fu, D. Y. Hong, Y. P. Liu, J. J. Wu, W. C. Hsu, Machinery Assoc Comp.Exploiting Vector Processing in Dynamic Binary Translation.In: 48th International Conference on Parallel Processing (ICPP). Univ Tsukuba, Ctr Computat Sci, Kyoto, JAPAN. 2019. 1-10. [doi:10.1145/3337821.3337844].

170. Ruoyu Zhou, George Wort, Márton Erdős, Timothy M. Jones.The janus triad: exploiting parallelism through dynamic binary modification.In: Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Providence, RI, USA: Association for Computing Machinery. 2019. 88–100. [doi:10.1145/3313808.3313812].

171. K. Jingu, K. Shigenobu, K. Ootsu, T. Ohkawa, T. Yokota.Directive-Based Parallelization of For-Loops at LLVM IR Level.In: IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). 2019. 421-426. [doi:10.1109/SNPD.2019.8935667].

172. Emilio G. Cota, Luca P. Carloni.Cross-ISA machine instrumentation using fast and scalable dynamic binary translation.In: Proceedings

of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Providence, RI, USA: Association for Computing Machinery. 2019. 74–87. [doi:10.1145/3313808.3313811].

173. Yuchuan Guo, Wuu Yang, Jiunnyeu Chen, Jenqkuen Lee.Translating the ARM Neon and VFP instructions in a binary translator. Software: Practice & Experience, 2016. 46(12). 1591–1615. [doi:10.1002/spe.2394].

174. Wang Wenwen, Wu Chenggang, Bai Tongxin, Wang Zhenjiang, Yuan Xiang, Cui Huimin.A Pattern Translation Method for Flags in Binary Translation. Journal of Computer Research and Development, 2014. 51(10). 2336-2347, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.2014.20130018].

175. Jie Tan, Jian-Min Pang, Shuai-Bing Lu.Using Local Library Function in Binary Translation.In: Current Trends in Computer Science and Mechanical Automation Vol.1. Warsaw, Poland: De Gruyter Open Poland. 2018. 123-132. [doi:10.1515/9783110584974-016].

176. Pang Janming Fu Liguo, Wang Jun, Zhang Jiahao, Yue Feng.Optimization of Library Function Disposing in Dynamic Binary Translation. Journal of Computer Research and Development, 2019. 56(8). 1783-1791, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.2019.20170871].

177. K. Chai, F. Wolff, C. Papachristou, Ieee.XBT: FPGA Accelerated Binary Translation.In: 73rd IEEE National Aerospace and Electronics Conference (NAECON). Dayton, OH. 2021. 365-372. [doi:10.1109/naecon49338.2021.9696395].

178. R. Wirsch, C. Hochberger.Towards Transparent Dynamic Binary Translation from RISC-V to a CGRA.In: 34th International Conference on Architecture of Computing Systems (ARCS). Electr Network. 2021. 118-132. [doi:10.1007/978-3-030-81682-7_8].

179. Tiago Knorst, Julio Vicenzi, Michael G. Jordan, Jonathan H. De Almeida, Guilherme Korol, Antonio C. S. Beck, Mateus B. Rutzig.An energy efficient multi-target binary translator for instruction and data level parallelism exploitation. Design Automation for Embedded Systems, 2022. 26(1). 55-82. [doi:10.1007/s10617-021-09258-6].

180. Saagar Jha. TSOEnabler Kernel extension that enables TSO for Apple silicon processes. 2020.https://github.com/saagarjha/TSOEnabler.

181. Simon Rokicki, Erven Rohou, Steven Derrien.Hardware-Accelerated Dynamic Binary Translation.In: 20th Conference and Exhibition on Design, Automation and Test in Europe (DATE). EPFL Campus, Lausanne, SWITZERLAND. 2017. 1062-1067. [doi:10.5555/3130379.3130632].

182. Simon Rokicki, Erven Rohou, Steven Derrien.Aggressive Memory Speculation in HW/SW Co-Designed Machines.In: Design, Automation & Test in Europe Conference & Exhibition (DATE). Florence, ITALY. 2019. 332-335. [doi:10.23919/DATE.2019.8715010].

183. wow64 implementation details. 2022.https://docs.microsoft.com/zh-cn/windows/win32/winprog64/wow64-implementation-details.

184. Microsoft. Windows on Arm. 2023 https://learn.microsoft.com/en-us/windows/arm/overview.

185. Wine. Wine. 2023.https://www.winehq.org/.

186. Run Android apps on the Android Emulator. 2018.https://tutorial.eyehunts.com/android/run-android-apps-android-emulator/.

187. Developers. Android Studio. 2022.https://developer.android.google.cn/studio/releases/emulator#support_for_arm_binaries_on_android_9_and_11_system_images.

188. Intel. Bridge Technology. 2023.https://www.intel.cn/developer/topic-technology/bridge-technology.html.

189. Intel. An Introduction to Celadon. 2022.https://www.intel.com/content/developer/topic-technology/open/celadon.

190. Nuno Paulino, João Canas Ferreira, João M. P. Cardoso.Improving Performance and Energy Consumption in Embedded Systems via Binary Acceleration: A Survey. ACM Comput. Surv, 2020. 53(1). Article 6. [doi:10.1145/3369764].

191. M. Panchenko, R. Auler, B. Nell, G. Ottoni.BOLT: A Practical Binary Optimizer for Data Centers and Beyond.In: 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2019. 2-14. [doi:10.1109/CGO.2019.8661201].

192. Sriraman Tallam.Propeller: Profile Guided Large Scale Performance Enhancing Relinker.In: Bay Area LLVM Developers' Meeting 2019.

193. Zou Wei, Gao Feng, Yan Yunqiang. Dynamic Binary Instrumentation Based on QEMU. Journal of Computer Research and Development, 2019. 56(4). 730, (in Chinese with English abstract). [doi:10.7544/issn1000-1239.2019.20180166].

194. Samuel Ginzburg, Mohammad Shahrad, Michael Freedman.VectorVisor: A Binary Translation Scheme for Throughput-Oriented GPU Acceleration.In: 2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023. 1017-1037.

195. Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, G. Vigna.SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis.In: 2016 IEEE Symposium on Security and Privacy (SP). 2016. 138-157. [doi:10.1109/SP.2016.17].

196. Vitaly Chipounov, Volodymyr Kuznetsov, George Candea.S2E: a platform for in-vivo multi-path analysis of software systems.In: Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems. Newport Beach, California, USA: Association for Computing Machinery. 2011. 265–278. [doi:10.1145/1950365.1950396].

197. Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, Taesoo Kim.QSYM: a practical concolic execution engine tailored for hybrid fuzzing.In:

Proceedings of the 27th USENIX Conference on Security Symposium. Baltimore, MD, USA: USENIX Association. 2018. 745-761.

198. Sebastian Poeplau, Aurelien Francillon, Usenix Assoc.Symbolic execution with SYMCC: Don't interpret, compile!In: 29th USENIX Security Symposium. Electr Network: USENIX Association. 2020. 181-198.

199. Mendel Rosenblum.VMware Virtual Platform Technology.In.: USENIX Association. 1999.

200. Avi Qumranet, Yaniv Qumranet, Dor Qumranet, Uri Qumranet, Anthony Liguori.KVM: The Linux virtual machine monitor. Proceedings Linux Symposium, 2007. 15.

201. Zheng Shan, Haoran Guo, Jianmin Pang.BTMD: A Framework of Binary Translation Based Malcode Detector.In: 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. 2012. 39-43. [doi:10.1109/CyberC.2012.16].

202. Michael Spreitzenbarth, Thomas Schreck, Florian Echtler, Daniel Arp, Johannes Hoffmann.Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques. International Journal of Information Security, 2015. 14(2). 141-153. [doi:10.1007/s10207-014-0250-0].

203. Mengchen Cao, Xiantong Hou, Tao Wang, Hunter Qu, Yajin Zhou, Xiaolong Bai, Fuwei Wang.Different is Good: Detecting the Use of Uninitialized Variables through Differential Replay.In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, United Kingdom: Association for Computing Machinery. 2019. 1883–1897. [doi:10.1145/3319535.3345654].

204. Peter Garba, Matteo Favaro.SATURN - Software Deobfuscation Framework Based On LLVM.In: Proceedings of the 3rd ACM Workshop on Software Protection. London, United Kingdom: Association for Computing Machinery. 2019. 27–38. [doi:10.1145/3338503.3357721].

205. Zeliang Kan, Haoyu Wang, Lei Wu, Yao Guo, Guoai Xu.Deobfuscating Android Native Binary Code.In: IEEE/ACM 41st International Conference on Software Engineering - Software Engineering in Practice (ICSE-SEIP). Montreal, CANADA. 2019. 322-323. [doi:10.1109/ICSE-Companion.2019.00135].

206. Ying Wang, Li-Ze Gu, Zhong-Xian Li, Yi-Xian Yang, Yi-Xian Yang.Protocol reverse engineering through dynamic and static binary analysis. The Journal of China Universities of Posts and Telecommunications, 2013. 20. 75-79. [doi:10.1016/S1005-8885(13)60217-4].

207. Scott Wasson. Nvidia claims Haswell-class performance for Denver CPU core. 2014.https://techreport.com/news/26906/nvidia-claims-haswell-class-performance-for-denver-cpu-core/.

208. Marcelo Brandalero, Muhammad Akmal Shafique, Luigi Carro, A. C. S. Beck.TransRec: Improving Adaptability in Single-ISA Heterogeneous Systems with Transparent and Reconfigurable Acceleration. 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019. 582-585. [doi:10.23919/DATE.2019.8715121].

209. Ding-Yong Hong, Shih-Kai Lin, Sheng-Yu Fu, Jan-Jan Wu, Wei-Chung Hsu.Enhancing transactional memory execution via dynamic binary translation. ACM SIGAPP Applied Computing Review, 2019. 19(1). 48–58. [doi:10.1145/3325061.3325065].

附中文参考文献:

[8]李剑慧,马湘宁,朱传琪.动态二进制翻译与优化技术研究.计算机研究与发展,2007,44(1):161-168.

[17] 胡伟武,汪文祥,吴瑞阳,王焕东,曾露,徐成华,高翔,张福新.龙芯指令系统架构技术.计算机研究与发展,2023,60(1):2-16.[doi:10.7544/issn1000-1239.202220196].

[18]李男,庞建民.基于中间表示规则替换的二进制翻译中间代码优化方法. 国防科技大学学报. 2021. 43(04):156-162

[24]刘晓楠.面向国产处理器的二进制翻译关键技术研究.博士,解放军信息工程大学.2014.

[41]马湘宁.二进制翻译关键技术研究.博士,中国科学院研究生院计算技术研究所.2004.

[92]胡伟武,靳国杰,汪文祥,张晓春,王焕东.龙芯指令系统融合技术.中国科学:信息科学.2015.45(4):459–479.[doi:10.1360/N112014-00300].

[120]傅立国,庞建民, 王军,张家豪,岳峰.二进制翻译正确性及优化方法的形式化模型.计算机研究与发展.2019.56(9):2001-2011 [doi:10.7544/issn1000-1239.2019.20180513].

[144]文延华,唐大国,漆锋滨.二进制翻译中的寄存器映射与剪裁的实现.软件学报.2009.20(zk):1-7.

[158]王军,庞建民,傅立国,岳峰,张家豪.一种高效解决间接转移的反馈式静态二进制翻译方法.计算机研究与发展. 2019. 56(4):742-754. [doi:10.7544/issn1000-1239.2019.20170412].

[163]王军,庞建民,傅立国,岳峰,单征,张家豪.二进制翻译中动静结合的寄存器分配优化方法.计算机研究与发展.2019.56(4):708-718.[doi:10.7544/issn1000-1239.2019.20170905].

[174]王文文,武成岗,白童心,王振江,远翔,崔慧敏.二进制翻译中标志位的模式化翻译方法.计算机研究与发展. 2014.51(10):2336-2347.[doi:10.7544/issn1000-1239.2014.20130018].

[176] 傅立国,庞建民,王军,张家豪,岳峰.动态二进制翻译中库函数处理的优化.计算机研究与发展,2019,56(8):1783-1791.[doi:10.7544/issn1000-1239.2019.20170871].

[193] 邹伟,高峰,颜运强.基于 QEMU 的动态二进制插桩技术.计算机研究与发展.2019.56(4):730-741.[doi:

10.7544/issn1000-1239.2019.20180166].