二进制翻译技术研究

武成岗、马湘宁、崔慧敏

1 引言

一种新处理器的流行，离不开相应软件的支持。开发新的处理器可能会因为失去相应软件的支持而影响其推广应用和市场前景；另一方面，得不到广泛应用和一定市场份额的处理器也很难得到丰富的软件支持。这种处理器和支持软件之间相互钳制的关系，既使得新处理器的设计不得不考虑兼容老处理器，也阻碍了新处理器的推出。在这种情况下，研究如何把支持老处理器的软件移植到新的处理器上，使新的处理器从诞生之初就有丰富的软件，不仅对软件重用有重大意义，更可以开阔处理器研发的思路，促进新处理器的创新。

一般有三种方法可以把老处理器上的代码移植到新处理器上[1]：

1. 在新处理器上提供专门的运行模式来执行老代码，如英特尔的安腾（Itanium）处理器专门设计了执行 x86 代码的硬件。
2. 把源程序重新编译到新的指令集。
3. 使用软件方法，解释或翻译应用程序。

第一种方法，显然无法利用新处理器的一些先进特性，失去了开发新处理器的意义，并且增加了新处理器的硬件复杂度，甚至还会影响原有代码的执行效率；第二种方法可以达到很好的效率，但并不总是可行，因为有些程序已经没有源代码，有些程序依赖于共享代码库，而这些共享代码以目标代码形式出现，不一定能得到源码，有些源程序语言没有编译到新指令集的编译器，此外操作系统的差异还可能使得只有修改源代码才能重新编译这些例程（比如与图形相关的代码）。

因此第三种方法，称之为二进制翻译（Binary Translation）应运而生。它是一种直接翻译可执行二进制程序的技术，能够把一种处理器上的二进制程序翻译到另外一种处理器上执行。它使得不同处理器之间的二进制程序可以很容易地相互移植，扩大了硬件/软件的适用范围，有助于打破前面提到的处理器和支持软件之间互相掣肘影响创新的局面。

2 二进制翻译技术概述

二进制翻译也是一种编译技术，它与传统编译的差别在于其编译处理对象不同。传统编译处理的对象是某一种高级语言，经过编译处理生成某种机器的目标代码；二进制翻译处理的对象是某种机器的二进制代码，该二进制代码是经过传统编译生成的，经过二进制翻译处理后生成另一种机器的二进制代码。按照传统编译程序前端、中端和后端的划分，我们可以理解为二进制翻译是拥有特殊前端的编译器。

2.1 二进制翻译方法分类及比较

基于软件的二进制翻译，可以分为三类：解释执行、静态翻译、动态翻译。图 2 展示了这三种方法，以及使用了该方法的翻译系统[1]。

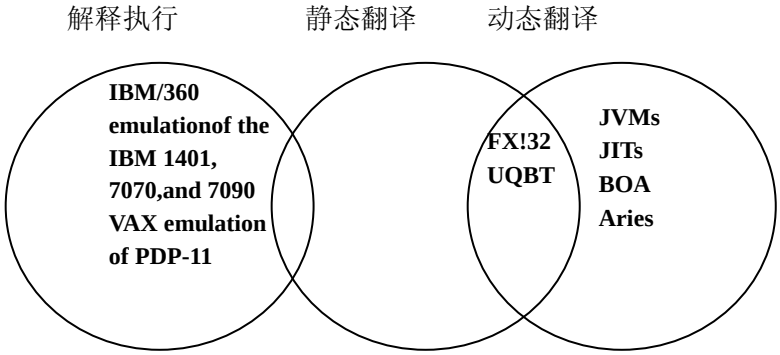

图 1　二进制翻译的三种方法

解释执行对源处理器代码中的每条指令实时解释执行，系统不保存也不缓存解释过的指令，不需要用户干涉，也不进行任何优化。解释器相对容易开发，比较容易与老的体系结构高度兼容，但代码执行效率很差[1] [14]。

静态翻译是在源处理器代码执行之前对其进行翻译，将源机器上的二进制可执行程序文件A完全翻译成目标机器上的二进制可执行程序文件B，然后在目标机上执行程序B。一次翻译的结果可以多次使用。静态翻译器离线翻译程序，有足够的时间进行更完整细致的优化，代码执行效率较高。然而，静态翻译器可能无法完整地翻译一个程序[30]，因而需要依赖解释器的支持[14]；而且静态翻译器需要终端用户的参与[1]，这给用户使用造成了很大不便。

动态翻译则在程序运行时对执行到的片断进行翻译，克服了静态翻译的一些缺点--如由于不能知道控制流中某点的寄存器或内存的值，因此不能实现代码挖掘；动态翻译还可以解决大部分实际情况中的自修改代码问题，而这在静态翻译是不可能的[14]；动态翻译可以利用执行时的动态信息来发掘静态编译器所不能发现的优化机会；动态翻译器对用户可以做到完全透明，无需用户干预[4] [9] [11] [19]。

下面我们用表格的形式总结一下上述三种翻译方法的优缺点[1] [14]：

	优点	**缺点**
解释执行	容易开发，不需要用户干涉，高度兼容	代码执行效率很差
静态翻译	离线翻译，可以进行更好的优化，代码执行效率较高。	依赖解释器、运行环境的支持，需要终端用户的参与，给用户使用造成了不便
动态翻译	无需解释器和运行环境支持，无需用户参与，可利用动态信息发掘优化机会	翻译的代码执行效率不如静态翻译高，对目标机器有额外的空间开销

表 1　三种翻译方法比较

2.2　二进制翻译系统框架

由于上述三类翻译方法各有所长，开发二进制翻译系统时一般会使用上述方法中的两种或三种。下面以解释执行和动态翻译相结合的二进制翻译系统框架为例，如图 2（许多真实系统都采用类似框架[4] [9] [11] [18]），简略描述一个二进制翻译系统的翻译执行过程。

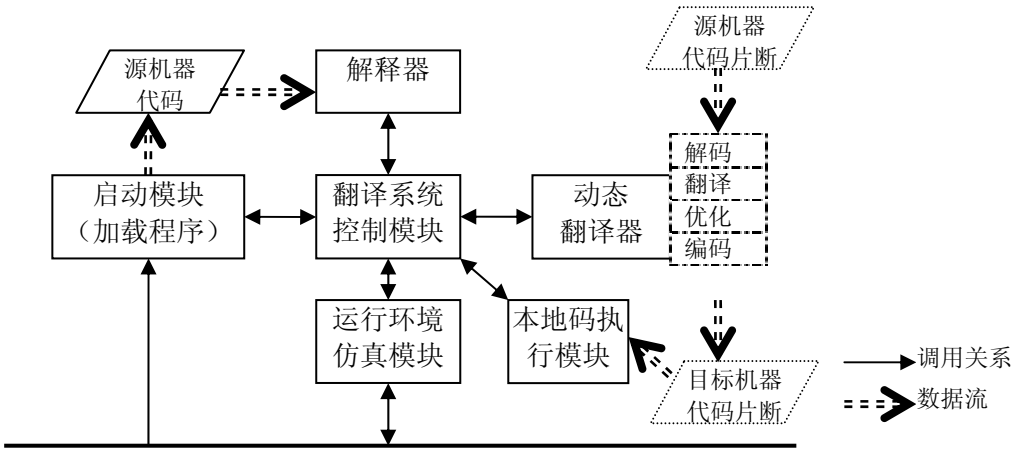

图 2 二进制翻译系统框架

一个解释执行和动态翻译相结合的二进制翻译系统，通常包括控制模块、启动模块、运行环境仿真模块、解释器、动态翻译器、本地码执行模块等主要模块。控制模块控制整个系统的翻译执行过程，从而可以使系统的工作对用户透明；本地码执行模块负责在翻译系统自身的执行和生成的目标机器代码执行之间的切换工作；运行环境仿真模块支持源机器代码调用、系统调用以及处理信号。

整个翻译执行过程如下：在目标机器平台下执行源机器代码时，嵌入操作系统的启动模块启动整个翻译系统，并将源机器代码加载到内存，然后控制模块启动解释器对源机器代码解释执行，并根据需要统计代码执行路径等信息。当某段源机器代码的执行达到一定热度，就启动动态翻译器翻译并对其进行简单优化，编码生成目标机器代码片断，此后再执行到该源机器代码片断时就不再解释执行，而是通过本地码执行模块来直接执行翻译生成的目标机器代码。

为了提高翻译后代码的效率，对于那些执行热度极高的目标机器代码片断，还需要对其进一步优化，这就会用到下文提到的动态优化技术。

2.3 动态优化技术简介

动态优化技术[23] [34] [39] [40] [41]既可以作为一种独立的代码优化技术，又可以作为二进制翻译所必需的后端优化器。而二进制代码的动态优化技术也需要解码器等模块的支持，因此动态优化技术与二进制翻译技术具有密切的关系。

动态优化技术在应用程序运行时刻对程序的信息进行收集和分析，并对程序的关键段进行必要的优化，从而提高程序的性能。由于在动态时刻对程序进行上述处理要耗费时间，所以动态优化必须及时发现程序的关键段，并采用快速而高效的分析、优化方式，以得到理想的优化效果。动态优化系统所用到的热路径选择技术可以用纯软件方式实现，若硬件结构可以提供必要的支持，则动态优化系统的效率可望得到显著提高[50]。

二进制翻译要兼顾正确性和高效率，在实现不同系列的CPU间的软件平滑移植目标时，动态翻译能够保证程序执行的正确性，但其运行会产生开销。而动态优化可以提高目标机代码质量补偿这些开销，这种需求推动了动态优化的研究；即便是同一个系列的芯片，由于不同“代”芯片之间的具体硬件资源不同，面向旧芯片的可执行代码虽然可以直接在新的芯片上执行[3]，但由于不能充分利用新的功能部件，程序的效率会受影响，因此也需要动态重优化技术对其加以改进。

3 有代表性的二进制翻译系统

目前，二进制翻译已经得到了广泛的重视和研究，一些有代表性的系统见表 2。

表 2 所列出的多数翻译器都与机器特性高度相关，重利用是非常困难的。其中 1994 年 AT&T 公司开发的 Flashport 二进制翻译器可以运用到多个源、目标平台，但不能完全自动化，需要专业用户通过图形用户界面（GUI）进行交互。昆士兰（Queensland）大学开发的 UQBT 以及 UQDBT，

都是基于对机器指令和操作系统属性描述的可变源、可变目标的二进制翻译器框架，可以看作是一个翻译器的生成器。

名称	研究单位	参考	目的	源平台	目的平台
Bergh et al (1987)	HP	[5]	最早的商用二进制翻译系统；用于软件模拟和目标代码翻译	(HP3000, MPE V)	(HP Precision Archi- tecture, MPE XL)
Mimic (1987)	IBM	[6]	对每条源机器指令代码扩展倍数为 4 的软件模拟器	IBM system /370	IBM RT PC
Accelera- tor (1991)	Tandem	[7]	将 CISC 移植到 RISC 的静态翻译器，采用解释器作为补充	TNS CISC	TNS /R
VEST, mx (1993)	Digital	[8] [4]	从 Digital 公司的 VAX 和 MIPS 到 64 位 Alpha 静态翻译器，采用解释器作为补充	(VAX, OpenVMS), (MIPS, Ultrix)	(Alpha, OpenVMS), (Alpha, OSF/1)
Flashport (1994)	AT&T	[14]	跨越多个源和目标平台的二进制翻译，需要人为干预	68x0 Mac, IBM system/360,370 ,380	Power Mac, IBM RS/6000 SPARC, HP, MIPS, Pentium
MAE (1994)	Apple	[14]	在 PowerMac 上开发的一个 Motorola 68000 解释器，后来改进成翻译器。	680x0	RISC-based Unix
Freeport Express (1995)	Digital		静态翻译器，采用解释器作为补充，翻译用户模式程序，32 位、64 位都可	(SPARC, SunOS 4.1.x)	(Alpha, OSF/1)
FX!32 (1996)	Digital	[2] [3]	流行的 x86 应用程序到 Alpha 的混合模拟器和二进制翻译器	(x86, Windows NT)	(Alpha, Windows NT)
Daisy (1996)	IBM	[9] [10]	利用二进制翻译调度 PowerPC 代码到超长指令字(VLIW)处理器，增加并行性	(PowerPC, UnixV)	VLIW
Aries (1999)	HP	[4]	解释和动态翻译相结合，简化了应用程序从 HP Precision Architecture, 到 IA-64 翻译；	HP Precision Architec- ture,	IA-64
BOA (1999)	IBM		动态翻译了 PowerPC 的整个系统，用简单的指令实现原来语义，简化硬件	(PowerPC, UnixV)	(PowerPC, UnixV)
UQBT、UQDBT	Queens- land 大学	[13] [14] [15] [16] [17] [18]	利用机器指令描述开发的可变源、可变目标的静态（动态）二进制翻译器	可变	可变
Code Morphing software (2000)	Transme- ta 公司	[19] [20]	动态翻译运行 x86 代码，翻译全部程序包括 Windows 操作系统	x86	Transmeta 芯片
Bintran	维也纳技术大学（奥地利）	[21] [38]	多源多目标的动态翻译器	可变	可变
QuickTrans -it	Transitive 公司			MIPS/PowerPC/x 86	Itanium/ x86/PowerPC /Opteron
DigitalBri dge	中科院计算所		动静结合的翻译	x86	MIPS

表 2　二进制翻译系统

由于静态二进制翻译器的局限性，所有的实用系统都不采用纯静态的翻译，而是选择动态模拟或动态翻译再加上动态优化的方式。这样就可以在保证程序能够正确执行的基础上，尽量提高效率，如 Daisy，Aries，BOA，Transmeta 等系统。而且动态翻译还可以对用户透明，从而无需用户对其过程进行干预。

下面着重分析几个有代表性的二进制翻译系统，并简单介绍 Dynamo 和 JIT 动态优化器。

3.1　特定目标的二进制翻译系统

DEC公司 1996 年研发的FX!32 系统[2] [3]，是一个剖析信息指导的（profile-directed）二进制翻译器，目的是为了将运行在x86/WinNT系统下的应用程序在Alpha/WinNT环境下运行。它结合静态翻译和动态解释，具有正确，高效而且透明的特点。FX!32 第一遍执行时，用解释器进行解释执行，并将执行路径等信息保存下来，由另一个后台程序根据这些剖析（profiling）信息进行静态翻译和优化，第二遍之后的执行，就开始部分使用翻译后的本地代码，部分仍然需要解释，执行中仍然要生成剖析信息，与前面的信息合并，再用于静态翻译优化。通过这个循环过程，随着程序执行次数增加，速度日益加快。FX!32 系统能够正确翻译WinNT下的Lotus、Excel、Word 等实用程序，且使翻译后的代码在Alpha500 上运行与在奔腾（Pentium）200 上运行性能相当，真正达到了实用。但FX!32 是运用静态翻译，因而第一遍执行速度较慢，且依赖纯静态的剖析和静态翻译不能根据程序执行时的变化动态优化代码。

HP公司 1999 年开发的Aries软件仿真器[4]，是一个基于软件的IA64[1]转化设备。该系统结合快速解释和动态翻译两种翻译手段，可以仿真PA-RISC全部指令集，无需用户干涉。Aries只动态翻译经常使用的代码，仿真过程结束后放弃所有翻译的代码，而不修改原来的应用程序。因此，动态翻译既可提供快速仿真，又不破坏仿真的PA-RISC应用程序的完整性。快速解释和动态翻译相结合，用户就可以在运行HP-UX 操作系统的IA-64 机器上透明、准确、有效地执行PA-RISC应用程序。此外，这种结合对指令集体系结构进行仿真由于只翻译少数代码，因而比其他仿真方法代价更低，翻译的代码比仿真的代码快，因而性能更高。Spec2000 整数测试表明，经Aries 转化的程序在Itanium2（1000M）机器上的运行时间约为本地码在PA-8700（750M）机器上运行的 50%，浮点性能略差。Aries系统也是实用的商业代码转化系统。

DigitalBridge 是由中科院计算所开发的一个动静结合的二进制翻译系统，它能够将 x86 的应用程序翻译到 MIPS 上执行。DigitalBridge 在静态时尽可能地对程序进行翻译，并由动态翻译器来弥补静态的不足，从而最大限度地提高翻译后程序的性能。在 DigitalBridge 系统中，为了提高性能，还采取了以下措施：1）根据 x86 体系结构浮点处理的特点，对浮点栈进行了高效模拟；2)根据 x86 应用程序中栈变量的特点，对局部变量进行识别和提升；3)根据多分支结构控制语句（switch-case）所对应的间接跳转，对跳转表进行识别和提升；4）根据 x86 的标志位特点，基于数据流和模式匹配对标志位进行处理；5）对跨平台的系统库函数调用进行了分类优化处理。在这些技术的支持下，DigitalBridge 能够使得 x86 的应用程序在 MIPS 平台上执行并获得良好的性能。

上述FX!32、DigitalBridge和Aries翻译系统都是在特定的操作系统环境下运行，着眼于应用程序的翻译；下面提到的Daisy、BOA、Transmeta系统则是对整个系统的翻译，包括应用程序、操作系统、以及其它特权级的指令[2]。

Daisy和BOA系统是IBM公司分别于 1996 年和 1999 年开发的，虽然都用到二进制翻译优化技术，动态解决了诸如精确中断、自修改代码和内存一致性等二进制翻译通常会遇到的难题，但这两个系统的研究开发目标并不相同。Daisy[9] [10]是用于仿真现存体系结构的二进制翻译系统，以使旧体系结构上现存的软件（包括操作系统内核）可以在超长指令字（VLIW[2]）体系结构下运行。

[1] IA64 是Intel与惠普共同设计的 64 位处理器架构，支持 64 位运算能力、64 位寻址空间和 64 位数据通路
[2] Very Long Instruction Word

VLIW结构设计简单而且指令发射率高，但却与现存的软件不兼容，使VLIW得不到真正使用，Daisy正是要解决这个问题。每当一段新的指令第一次执行，这些代码就被驻留在只读内存中的虚拟机监视器翻译成VLIW原语，并行化且保存在旧的体系结构看不到的内存中，以后再执行这段代码时就无需翻译。Daisy实现了对于PowerPC体系结构的动态并行化算法，以较低的翻译开销，获得了较高的指令级并行度。

BOA动态翻译器系统[11] [12]的目标是简化硬件，通过结合二进制翻译和动态优化，填补PowerPC RISC指令集和更简单的硬件原语之间的语义差别。BOA系统关心的不是每条指令的周期数目（CPI[3]）最小化，而是希望通过简化的硬件指令，极大地提高处理器频率。BOA系统除了动态地解决二进制翻译中存在的精确中断和自修改代码问题，还通过在解释过程中收集轮廓(profile)信息，帮助生成路径(trace)，将一条路径上的代码放于内存连续位置，提高了指令缓存(cache)命中率，有助于迅速取址。BOA还进行了优化调度，从而提高了程序并行性，并解决了由调度产生的访存一致性问题。BOA为取得最大的指令级并行（ILP[4]），同时进行乱序调度、优化和寄存器分配。

Transmeta公司 2000 年开发的代码变换（Code Morphing）软件[19] [20]，使自己研发的芯片能够兼容x86 的软件，利用二进制翻译技术开创了一个新的软、硬件开发模式。Transmeta公司生产的crusoe芯片，是由逻辑上被Code Morphing软件裹着的硬件引擎构成的。该引擎是个VLIW 的CPU，其设计的出发点只是利于低功耗实现。Code Morphing软件能够把x86 指令变成VLIW 指令，从而使x86 程序仿佛直接运行在x86 硬件上。Crusoe 芯片与Code Morphing软件的结合说明微处理器可以被当做软硬件的混合体来实现。软件部分的升级独立于芯片。硬件设计与系统和应用软件分开使得硬件设计师更新设计时不用担心影响遗留的软件。Code Morphing 软件本质上是一个动态翻译系统。它驻留在ROM[5] 之中，是处理器启动后第一个执行的程序。Code Morphing软件支持x86 代码，并且是用VLIW 指令写的唯一一个程序。由于Code Morphing 软件把x86 程序（包括BIOS[6] 与操作系统）与它本身的指令集隔开，所以改动它本身的指令集一点也不影响x86 软件。唯一要移植的程序是Code Morphing软件本身，这个工作由Transmeta 完成，并且随每次体系结构改变做一次就够了。Code Morphing 软件一直在透明地重编译和优化运行的x86 代码。这种技术的代价是占用处理器运行时间，因而为了能具有好的系统性能，需要仔细设计Code Morphing软件来达到最大的效率，最低的开销。

3.2 可变源和目标的二进制翻译系统

昆士兰大学先后研究开发了可变源和目标的静态（UQBT[13] [14] [15]）和动态（UQDBT[16] [17] [18]）二进制翻译系统框架，引导了二进制翻译研究中一个新的研究方向。UQBT框架根据不同的二进制文件格式描述文件，自动生成文件编解码器；根据不同的编解码描述文件自动生成指令编解码器；还要根据不同的语义描述文件，自动生成语义抽象转换器，从而使机器不相关部分的工作分离出来，给二进制翻译器编写者提供可重用的代码，使之仅需在独立于机器的抽象表示层上研究不同的变换优化算法。如果要移植到其它机器，则仅需重写描述文件即可。目前UQBT仅处理用户代码（应用程序代码），而非内核代码或动态连接库。UQBT使用两种中间表示，机器相关的寄存器传输列表（RTLs[7]）和机器无关的高层寄存器传输语言（HRTL[8]）。将源程序通过文件解码、指令解码生成RTLs，然后语义抽象到HRTL高级表示，在此基础上即可进行机器无关的分析和转换，最后再由HRTL生成目标机器的二进制代码。UQBT项目证明了通过使用描述，低代价支持不同机器创建可适应性二进制翻译环境的可行性。写描述比写与机器相关的部分源代码无论是时间还是代码数量都减少了很多，而且开发者还可以重用本系统提供的机器无关的分析。

UQBT在可重用二进制翻译器框架的搭建上是非常成功的，这在他们后期开发其他类型编译器前端和后端的实践中得到证明，利用这种技术只需耗费较少的人力和时间就完成了其他源、目标

[3] Cycles Per Instruction，周期/指令
[4] Instruction Level Parallelism
[5] Read-Only Memory，只读存储器
[6] Basic Input Output System基本输入/输出系统
[7] Register Transfer List
[8] Higher-Level Register Transfer Language

机器翻译系统的实现工作。就执行效率而言，UQBT采用现有gcc[9]或cc[10]编译器的后端优化器，性能与本地码相当，而UQDBT除了寄存器作了简单对应优化，没做其它优化，速度只有原来的 1/6 到一半。目前UQBT和UQDBT系统能够通过的测试例基本都是小程序，因而只是一个很好的实验平台而不是实用系统。

奥地利一所大学正在研究开发机器可适应（machine-adaptable）动态二进制翻译系统Bintran，其最终目标是希望在不同机器描述协助下，能够实现所有的CISC、RISC以及VLIW体系结构之间的代码翻译。Bintran系统最主要的部分是翻译器的生成器，它能够根据源、目标机器的机器描述产生一个C语言写的翻译器。此外还包含机器无关的调度器和寄存器分配器。系统调用接口和内存管理模块主要是与操作系统相关，而与指令集体系结构的相关度较小。最后，Bintran系统还包括若干体系结构相关部分：目标汇编码写的用于从调度器到翻译生成代码之间切换的切换代码；解决一些特殊情况的代码（如变长指令、目标机器寄存器少于源机器寄存器、不支持长立即数、以及条件码等情况）。与UQDBT使用三种描述语言不同，Bintran仅使用一种描述语言来描述源和目标机器的语法语义，而加载不同的源二进制文件是手写的。目前Bintran仅支持静态连接的ELF[11]二进制码，模拟了Linux系统调用接口，支持PowerPC到Alpha体系结构，x86 到Alpha体系结构的翻译，根据文献提供的测试数据可知，其运行时间大概是本地码的 2.6－5.3 倍。

Transitive公司拥有的Dynamite[22]软件技术，是英国曼彻斯特大学的超过 20 人年的研究成果。该技术可以使本地码和非本地码的程序都能无缝透明的执行。Dynamite软件分为三部分：原二进制码的解码（前端）、核心优化以及目标二进制码生成（后端）。前端和后端都是可以替换的，因而能支持多种处理器。这种结构还可以同时支持多个前端，因此允许多种二进制码流输入。前端接收源处理器的二进制码，翻译成中间表示（IR[12]），中间表示代码流用有向无环图（DAG[13]）表示。核心优化是在Dynamite设计的中间表示上进行的，使翻译的程序可以以相等甚至高于本地码的速度执行。优化包括：识别基本块和基本块组（根据基本块间高频率的交互和跳转生成）；识别热点：当基本块和基本块组执行频率高于其它部分代码时，被判为热点，对其高度优化，进行分支预测，代码重排；死代码删除：根据程序上下文，生成特殊实例代码，删除不需要的指令。后端主要是生成目标二进制指令，同时还根据目标机的特点进行后端优化，如指令级并行，代码调度。在此技术的基础上Transitive公司开发了QuickTransit翻译系统。

可变源和目标的二进制翻译系统目标是提供一个可重用的框架，提供某种中间表示并在上面进行优化，在开发某个具体的二进制翻译器时，可以重利用这些公共的代码，加速定制的二进制翻译器的开发。然而在 QuickTransit 翻译系统之前，这些可变源和目标的二进制翻译系统，仅在一定范围的源和目标处理器中通过了一些简单测试程序，还没有一个真正实用的系统。

3.3 动态优化器

HP公司开发的Dynamo[23] [36]是一个动态优化器的原型。它的输入是本地二进制可执行代码，通过解释执行并观察程序的行为而不需要任何采样代码，不需要对代码进行预分析，也不需要为以后的执行写出信息。解释执行程序时收集的剖析信息帮助动态选择频繁执行的热路径（hot trace），然后对这些热路径运用代码重布局、消除间接转移、以及一些常规优化技术，再将优化生成的代码存放在一个软件缓存中，当再执行到这些路径的时候就不解释而直接执行软件缓存中优化后的代码，从而使程序的执行效率得到大幅度提升。

Java编译器将java源代码编译成平台无关的Java 字节码（bytecode[14]）格式，然后将其分发到各种平台，由Java虚拟机（JVM[15]）实时解释执行。在高性能实现的JVM中，Just-In-Time（JIT）编译器实时地将Java字节码翻译成本地码，来减少解释执行的开销。由于翻译本身在程序执行期间进行，编译时间成为程序执行时间的一部分，因而编译中的优化需要考虑动态优化的特点，即需要在优化的代码质量和编译时间之间进行权衡。许多Java虚拟机中都采用了JIT编译，如Sun

[9] GNU C Compiler。GNU是 1984 年发起的一项计划，目的是开发和UNIX完全一致的免费操作系统
[10] C Compiler
[11] Executable and Linking Format，可执行与可链接格式
[12] intermediate representation
[13] Directed Acyclic Graphic
[14] 由Java编译器生成的跟机器相关代码，由Java解释器执行
[15] Java Virtual Machine

公司商业虚拟机JDK、汉城国立大学和IBM合作开发的开放源码Latte VM[26]、Intel的开放源码MRL VM[24] [25]。

Howard Chen等在[51]中提供了一个基于ORC[16]的动态优化框架，以在现代体系结构上用很小的开销来对目标代码进行动态优化。这一系统首先收集程序执行的性能信息，并基于这些信息针对目标机实施优化，最终用优化之后的代码片断来替代原代码片断的执行。

需要注意的是，虽然动态优化可以很好地提高动态翻译生成代码的效率，但我们也不能忽视其自身的开销。因而用较低的动态优化开销换取关键路径上的高效代码，是动态优化系统大幅度提高代码执行效率的关键。

4 二进制翻译技术需要解决的问题

所有的计算机都是图灵机，因此在一台机器上做的任何计算都能在另一台机器上仿真。构建有效的仿真源机器本身已是一个艰巨的任务，而二进制翻译要做的还不仅仅是仿真源机器，它还希望在目标机器上仿真源机器的代码时，执行效率能够达到或超过源机器本地代码执行效率，这需要对以下几个问题进行研究，从而既可以全面了解一个实用的二进制翻译优化系统需要解决的问题，又可以指导新机器的研发。

4.1 存储器相关问题

翻译时源机器的寄存器内容是必须保存的，这可以通过映射到目标机器的寄存器实现，假如目标机器的寄存器数少于源机器，则源机器中的一些寄存器内容就必须存在存储器中，这样，存取就需要花很大开销。保存在专用寄存器中的系统状态在目标机器中也必须保持。比如，要是源机器中有分段寄存器，在目标机器中就必须建立其镜像。另外还必须考虑保存条件码的标志寄存器，一种简单办法是像源机器一样在目标机器中保留它们，或者用软件的方法模拟并简化其变化[29]。

存储器映像I/O端口给整个系统的翻译带来了麻烦[1]。虽然以解释器为代表的各种仿真环境对这类问题提供了一种解决方式，但是解释器的效率太差，这是一个实用的二进制翻译系统所不能接受的。

数据表示的大尾端/小尾端是数据映射的一个问题[15]。如x86 平台采用的是小尾端的形式，MIPS平台既支持小尾端，也支持大尾端，而一些平台则只支持大尾端的形式。在这二者间进行翻译时，就需要根据数据的表示形式，在必要的时候，将表示一个数据对象的多个内存单元变换次序等，以保证在翻译后的二进制代码中的数据解读准确。

4.2 体系结构相关问题

许多机器都有与访存相关的一些具有原子性操作的指令，也就是说当执行这些指令时其它处理器不能访问存储器。在另一种结构的机器上仿真这种语义比较困难。不可中断性导致的问题与原子性相类似。某些体系结构的机器，比如IBM S/390，含有需要很多周期才能执行完的复杂指令，而且要求要么执行完，要么根本不执行[37]。在新体系结构的机器可以先试着运行这些指令的目标指令，如果没有副作用，就可以保证实际运行时正常工作[1]。

相对于不可中断性，更普遍的问题是处理老机器中的精确异常。比如，如果读存储器时发生了异常，老机器的中断处理程序会认为加载（load）指令前的所有指令都执行完毕，而加载指令后的所有指令都没执行完。如果目标代码进行了重排序，精确异常问题就非常棘手。通常精确异常在静态下无法解决，即使是动态也不易解决[31] [32] [9] [11] [19]。

4.3 代码挖掘问题

在冯·诺伊曼结构机器中，代码和数据以相同形式表示。因此前端解码器准确地挖掘出源机器的代码很困难。一般的方法是从代码入口点开始，沿着可到达的路径解码[34] [33]，但对静态翻译

[16] open research compiler开放源码编译器

来说，由于间接跳转和间接调用的存在，使这种方法失去效力。文献[30]采用了基于分割和表达式替换方法恢复跳转表目标地址的技术，可以克服这一困难。

与挖掘源机器代码相关的两个问题是自引用代码问题（self-referential code ，即代码引用它本身，比如求检验和[1] [28]）；以及自修改代码(self-modifying code)问题。解决自修改代码问题，首先是要能够发现自修改代码，这对一个纯静态翻译器来说是不可能的，对动态翻译器来说也不容易[1]。新机器如果能够提供一些手段监视修改代码[9] [11] [19]，就可以置对应的本地码为无效[28]。

4.4 效率问题

在二进制翻译系统中，尤其是动态翻译，一些工作是在翻译执行源二进制代码的过程中完成的，翻译系统本身的开销也被计算在源二进制代码的执行开销中。因此，提高二进制翻译器的效率要从两方面入手：首先是研究有效的分析、优化方法，提高翻译生成的目标二进制代码的质量；同时，要降低二进制翻译系统本身的开销，也就是降低翻译优化算法自身的复杂度。如果硬件系统能提供必要的支持，将有助于达到上述目的。

研究代码的重用率也可以提高二进制翻译的效率。如果翻译生成的代码可以频繁执行，就可以抵消翻译和优化源机器代码的代价[23] [27]，这需要做有效的剖析[43] [44] [45] [46] [47]。有效的代码缓存管理，也可以提高翻译系统效率。代码缓存总是有限的，当缓存被填满或者某些生成的代码变为无效时（如自修改代码），都需要高效处理[27] [48] [49]。

4.5 实时性问题

某些系统的代码有执行的时间限制，或者说必须以某种速率执行，这对二进制翻译来讲可能会有问题，因为二进制翻译系统执行程序快慢是变化的，其速度完全取决于将要执行的某段代码是否已经被翻译过。因此，在某些情况下，实时性的代码需要用启发式算法提前得知，并提前进行翻译[27] [28]。

4.6 运行环境仿真

运行环境仿真用于处理系统服务，主要是系统调用和信号传递。它捕获源机器应用程序做出的系统调用，调用相应的仿真例程。在应用程序级的二进制翻译系统中一般采用直接调用目标机器本地系统调用的方法，由于源和目标机器的应用程序二进制接口（ABI,Application Binary Interface）不同，在调用目标机器本地调用前还需要特殊处理[2] [3] [4]，若是跨操作系统的翻译，两个不同操作系统的系统调用可能无法完全一一对应，情形就更加复杂。同时对于操作系统发给应用程序的同步、异步信号，也应该能提供所需的特殊处理。对于能翻译包括操作系统在内的所有程序的全系统翻译器，则需要处理全部的处理器特权指令，见文献[9] [11] [19]。

5 总结和展望

二进制翻译及相关优化技术的研究有重要的意义。首先，二进制翻译有助于解决遗产代码问题，从而促进处理器设计创新[2] [3] [4] [5] [6] [7] [8]，这也是产生二进制翻译的源动力。其次，二进制翻译可以降低硬件复杂度，从而减少功耗，提高主频[11] [12]。第三，二进制翻译及其动态优化技术有助于提高程序性能[23]。动态优化技术可以作为传统静态编译器生成的代码的重优化器，在程序执行时发掘静态编译器无法发掘的优化机会，进一步提高程序的性能。第四，二进制翻译及其动态优化技术可以满足某些网络应用的需要。动态优化可以大大提高JAVA等语言编写的网络程序的性能[24] [25] [26]。第五，二进制翻译可以降低软件移植和维护费用[14]。

本文重点探讨了二进制翻译的特点，剖析了各类具有代表性的二进制翻译系统，并简要介绍了与二进制翻译相关的动态优化技术。目前为止，虽然在这个方向上做了大量的研究工作，但是还有很多问题值得深入探讨：1.对虚拟指令集（如Java虚拟机）的翻译优化[1]；2.使用更有效的剖析技巧来反映程序执行状态的变换[1]；3.支持不可中断性、原子性以及多平台的研究[1]；4.为二进制翻译提供有效的开发和调试支持[28]；5.对全系统二进制翻译的内存管理，启动代码和BIOS

代码的翻译[27]。

二进制翻译及相关优化技术是现代编译技术研究的热点之一，也是非常有前景的研究方向。而且这方面的技术和系统可望对正在开展的很多方向性研究，如网格技术、CPU 设计、移动计算和网络计算机等，提供重要的支持。

参考文献：

[1] Erik R.Altman, David Kaeli and Yaron Sheffer, “Welcome to the Opportunities of Binary Translation,” Computer, Vol 33, No 3, March 2000, IEEE Computer Society Press, pp 40-45.

[2] Anton Chernoff, Mark Herdeg, Ray Hookway, Chris Reeve, Norman Rubin, Tony Tye, S. Bharadwaj Yadavalli, and John Yates, “FX!32: a Profile-Directed Binary Translator”, IEEE Micro, vol. 18, no.2, 1998.

[3] R. J. Hookway and M. A. Herdeg, “Digital FX!32: Combining Emulation and Binary Translation,” Digital Technical J., Vol. 9, No.1, 1997, pp. 3-12.

[4] Cindy Zheng and Carol Thompson, “PA-RISC to IA-64:Transparent Execution,No Recompilation,” Computer, Vol 33, No 3, March 2000, IEEE Computer Society Press, pp 47-52.

[5] A. Bergh, K. Keilman, D. Magenheimer, and J. Miller, “HP3000 emulation on HP precision architecture computers,” Hewlett-Packard Journal, pages 87-89, Dec.1987.

[6] C. May, “MIMIC: A fast System/370 simulator,” In Proceedings SIGPLAN'87 Symposium on Interpreters and Interpretive Techniques, pages 1-13, June 1987.

[7] K. Andrews and D. Sand, “Migrating a CISC computer family onto RISC via object code translation,” In Proceedings ASPLOS V, pages 213-222, Oct. 1992.

[8] R. Sites, A. Chernoff, M. Kirk, M. Marks, and S. Robinson, “Binary translation (for the Alpha AXP architecture)”, Commun. ACM,36(2):69-81, Feb. 1993.

[9] K.Ebcioglu and E.Altman, “DAISY:Dynamic Compilaton for 100 Percent Architectural Compatibility,” Proc.ISCA24, ACM Press, New York, 1997, pp.26-37.

[10] K.Ebcioglu et al., “Execution-Based Scheduling for VLIW Architectures,” Proc. Europar99, Lecture Notes in Computer Science 1685, Springer Verlag, Berlin, 1999, pp.1269-1280.

[11] Michael Gschwind et al., “Dynamic and Transparent Binary Translation,” Computer, Vol 33, No 3, March 2000, IEEE Computer Society Press, pp 54-59.

[12] Michael Gschwind, Erik Altman, “Inherently Lower Complexity Architectures using Dynamic Optimization,” Proc. Workshop on Complexity Effective Design in conjunction with ISCA-2002, Anchorage, AK, May 2002.

[13] C.Cifuentes and M.Van Emmerik, “UQBT: Adaptable Binary Translation at Low Cost,” Computer, Vol 33, No 3, March 2000, IEEE Computer Society Press, pp 60-66.

[14] C.Cifuentes and V.Malhotra, “Binary Translation: Static, Dynamic, Retargetable?,” Proceedings International Conference on Software Maintenance. Monterey, CA, Nov 4-8 1996. IEEE-CS Press. pp 340-349.

[15] C.Cifuentes, M.Van Emmerik, D.Ung, D.Simon and T.Waddington, “Preliminary Experiences with the Use of the UQBT Binary Translation Framework,” Proceedings of the Workshop on Binary Translation, NewPort Beach, Oct 16, 1999. Technical Committee on Computer Architecture Newsletter, IEEE-CS Press, Dec 1999, pp 12-22.

[16] C Cifuentes, B Lewis and D Ung, “Walkabout - A Retargetable Dynamic Binary Translation Framework,” Fourth Workshop on Binary Translation, Sep 22, 2002, Charlottesville, Virginia.

[17] D Ung and C Cifuentes, “Optimising Hot Paths in a Dynamic Binary Translator,” Second Workshop on Binary Translation, Oct 19, 2000, Philadelphia, Pennsylvania.

[18] D Ung and C Cifuentes, “Machine-Adaptable Dynamic Binary Translation,” Proceedings of the ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization, Boston, USA, Jan 2000, ACM Press, pp 30-40.

[19] Alexander Klaiber, “The Technology behind Crusoe Processor,” Transmeta technology report, Jan. 2000. pp 3-12.

[20] Transmeta Corporation 3940 Freedom Circle, http://www.transmeta.com

[21] Mark Probst. Fast machine-adaptable dynamic binary translation. In Proceedings of the Workshop on Binary Translation 2001, September 2001.

[22] http://www.transitives.com/pressroom.htm#briefs

[23] Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia. “Dynamo: A Transparent Dynamic Optimization System”, In Proceedings of the ACM SIGPLAN '2000 conference on Programming language design and implementation, PLDI'2000, June, 2000

[24] M. Cierniak, G. Lueh, and J. Stichnoth, “Practicing JUDO: Java Under Dynamic Optimizations,” In Proceedings

of the ACM SIGPLAN '00 Conference on Programming Language Design and Implementation, October 2000.

[25] Intel ORP project, http://www.intel.com/research/mrl/orp.

[26] Seoul National University, IBM, http://latte.snu.ac.kr, 1999

[27] Altman ER. Ebcioglu K. Gschwind M. Sathaye S., “Advances and future challenges in binary translation and optimization”, In Proceedings of the IEEE, vol.89, no.11, Nov. 2001, pp.1710-22. Publisher: IEEE, USA.

[28] Y.N. Srikant, P. Shankar, “The Compiler Design Handbook, Optimizations and Machine Code Generation”, Chapter 19, CRC PRESS.

[29] M Srinivasan. Method and apparatus for emulating status flag. US Patent 5774694, 1998

[30] C.Cifuentes and M.Van Emmerik, “Recovery of Jump Table Case Statements from Binary Code,” Proceedings of the International Workshop on Program Comprehension, Pittsburgh, USA, May 1999, IEEE-CS Press, pp 192-199.

[31] Michael Gschwind and Eric R. Altman, “Optimizing and Precise Exceptions in Dynamic Compilation”, Second Workshop on Binary Translation Held in PACT 2000.

[32] Michael Gschwind and Eric R. Altman, “Precise Exception Semantics in Dynamic Compilation”, Proceedings of the Symposium on Compiler Constructions, April 2002.

[33] C. Cifuentes and K. Gough. “Decompilation of binary programs”, Software – Practice and Experience, 25(7):811–829, July 1995.

[34] R. Sites, A. Chernoff, M. Kirk, M. Marks, and S. Robinson. “Binary translation”, Commun. ACM, 36(2):69–81, Feb. 1993.

[35] Leonid Baraz, Tevi Devor, Orna Etzion, Shalom Goldenberg, Alex Skaletsky, Yun Wang and Yigal Zemach, “IA-32 Execution Layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium?based systems”, Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36), IEEE-CS Press.

[36] Bala, V., Duesterwald, E., and Banerjia, S. 1999. “Transparent dynamic optimization: The design and implementation of Dynamo”, Hewlett Packard Laboratories Technical Report. HPL-1999-78. June 1999. Dallas. 173-181.

[37] Gschwind M. Ebcioglu K. Altman E. Sathaye S, “Binary translation and architecture convergence issues for IBM system/390”, Conference Proceedings of the 2000 International Conference on Supercomputing. ACM. 2000, pp.336-47. New York, NY, USA.

[38] Probst M. Krall A. Scholz B, “Register liveness analysis for optimizing dynamic binary translation”, Proceedings Ninth Working Conference on Reverse Engineering WCRE 2002. IEEE Comput. Soc. 2002, pp.35-44.

[39] D. Bruening, E. Duesterwald and S. Amarasinghe, Design and Implementation of a Dynamic Optimization Framework for Windows, in Proceedings of the 4th Workshop on Feedback-Directed and Dynamic Optimization, December 2001

[40] W. Chen, S. Lerner, R. Chaiken and D. Gillies, Mojo: A Dynamic Optimization System, in Proceedings of the 3rd Workshop on Feedback-Directed and Dynamic Optimization, December 2000.

[41] D.H. Friendly, S.J. Patel and Y.N. Patt, Putting the Fill Unit to Work: Dynamic Optimizations for Trace Cache Microprocessors, in Proceedings of the 31st Annual International Symposium on Microarchitecture(MICRO-31), 1998, pp. 173-181.

[42] Sites RL. Chernoff A. Kirk MB. Marks MP. Robinson SG, “Binary translation”, Digital Technical Journal, vol.4, no.4, 1992, pp.137-52. USA.

[43] L. Anderson, M. Berc, J. Dean, M. Ghemawat, S. Henzinger, S. Leung, L. Sites, M. Vandervoorde C. Waldspurger and W. Weihl, Continuous Profiling: Where Have All the Cycles Gone, in Proceedings of the 16th ACM Symposium of Operating Systems Principles, 14, 1997.

[44] T. Conte, B. Patel, K. Menezes and J. Cox, Hardware-based profiling: an effective technique for profile-driver optimization, Int. J. Parallel Programming, 24, 187-206, April 1996.

[45] X. Zhang, Z. Wang, N. Gloy, J. Chen and M. Smith, System Support for Automatic Profiling and Optimization, in Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997, pp. 15-26.

[46] T. Ball and J. Larus, Efficient Path Profiling, in Proceedings of the 29th Annual International Symposium on Microarchitecture (MICRO-29), Paris, 1996, pp. 46-57.

[47] O. Taub, S. Schechter and M.D. Smith, Ephemeral Instrumentation for Lightweight Program Profiling, Technical Report, Harvard University, 2000.

[48] Kim Hazelwood, Michael D. Smith. “Generational Cache Mnagement of Code Traces in Dynamic Optimization Systems.” Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36-2003), 0-7695-2043-x/03 $17.00 © 2003 IEEE

[49] Kim Hazelwood, Michael D. Smith. “Code Cache Management Schemes for Dynamic Optimizers.” Proceedings of the Sixth Annual Workshop on Interaction between Compilers and Computer Architectures

(INTERACT.02), 0-7695-1534-7/02 $17.00 © 2002 IEEE

[50] Howard Chen, Wei-Chung Hsu, Jiwei Lu, Pen-Chung Yew, Dong-Yuan Chen. “Dynamic trace selection using performance monitoring hardware sampling”, Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, San Francisco, California, 2003, p79-90

[51] Howard Chen, Jiwei Lu, Wei-Chung Hsu, and Pen-Chung Yew, “Continuous Adaptive Object-Code Re-optimization Framework”, Ninth Asia-Pacific Computer Systems Architecture Conference (ACSAC 2004), pp. 241 - 255

作者简介：

武成岗： 中科院计算技术研究所助理研究员，博士，研究方向高级编译技术，曾经发表论文 18 篇，申请并被受理的专利 8 项，获军队科技进步二等奖一次。

马湘宁： 中科院计算技术研究所博士生，研究方向高级编译技术。

崔慧敏： 中科院计算技术研究所研究实习员，硕士，研究方向高级编译技术。