doi:10.3772/j.issn.1002-0470.2021.01.004

MIPS 安卓平台上 ARM 二进制翻译系统①

赵保华②*,**,*** 安宁钰③**,*** 徐哲冲**** 杜安利**** 苏 涛****

(*北京工业大学信息学部 北京 100124)

(**全球能源互联网研究院有限公司 北京 102209)

(***电力系统人工智能(联研院)国家电网公司联合实验室 北京 102209)

(****龙芯中科技术有限公司 北京 100095)

摘 要 无内部互锁流水级的微处理器(MIPS)是重要的处理器架构,安卓是目前主流的移动终端操作系统。在 MIPS 架构处理器上运行安卓操作系统可以有效拓展使用领域,但存在的问题是调用高级精简指令集处理器(ARM)架构本地库的安卓应用程序不能运行,即存在应用不兼容问题。本文提出了一种动态库跨平台二进制兼容模型,以及通用的跨平台二进制翻译系统的架构,并在 MIPS 安卓平台上设计实现了 ARM 二进制翻译系统。该系统能够进行 ARM 动态库的跨平台加载,并采用动态二进制翻译,将 ARM 动态库中的二进制指令翻译成为 MIPS 架构的二进制指令,从而能够在 MIPS 架构处理器上执行。实验结果表明,该系统可以运行调用 ARM 本地库的安卓应用程序,解决了 MIPS 平台安卓应用的兼容性问题。本文工作对跨平台二进制翻译系统的研究具有重要参考价值。

关键词 二进制翻译;无内部互锁流水级的微处理器(MIPS);高级精简指令集处理器(ARM);Android

0 引 言

无内部互锁流水级的微处理器(microprocessor without interlocked piped stages,MIPS)架构是 1981 年由斯坦福大学开发的简洁、具有高度可拓展性的精简指令集计算机(reduced instruction set computer,RISC)架构,MIPS 公司的 R 系列处理器就是在这个架构的基础上研制出来的微处理器。MIPS 架构处理器被广泛应用于游戏机、机顶盒、网络路由器,以及其他嵌入式设备和通设备中。我国自主研制的龙芯 CPU 基于开放授权的 MIPS 架构[1-3],并在此基础上扩展形成了自己的指令架构。目前,龙芯 CPU 已经广泛地应用于办公、卫星导航、信息安全等关键领域。

目前安卓(Android)操作系统是移动终端上的主流操作系统。安卓应用是采用谷歌公司的软件开发工具包安卓 SDK(standard development kit)和 NDK(native development kit)开发的[4]。仅使用 SDK 开发的应用可以凭借 Java 语言的通用性在所有安卓虚拟机支持的架构(目前包括 ARM、X86 和 MIPS)上正常运行。然而使用 NDK 开发的安卓应用会在程序包中生成动态共享库文件,这种文件包含有与相应体系结构相对应的机器代码片段,由安卓虚拟机通过 Java 本地接口 JNI(Java native interface)调用[5],只能在相应架构的硬件设备上运行。由于采用高级精简指令集处理器(advanced RISC machine, ARM)架构处理器在移动设备市场上垄断性的地位,大部分采用 NDK 开发的应用只支持

① 国家自然科学基金(61521092)和国家电网有限公司总部科技(SGGROOOOJSJS18002031)资助项目。

② 男,1984 年生,博士生;研究方向:计算机科学与技术;E-mail:zbh1984_1@126.com

③ 通信作者,E-mail:anningyu@geiri.sgcc.com.cn

(收稿日期:2019-12-11)

ARM 架构,导致了兼容性问题。

二进制翻译是解决不同指令系统兼容的重要技术[6-8]。二进制翻译技术通过在宿主机上用软件模拟出一个目标机指令系统兼容的 CPU,从而在宿主机上执行客户机的二进制代码,达到兼容的目的。如在 MIPS 计算机上模拟 ARM 指令系统,从而实现 ARM 兼容。但是二进制翻译存在较严重的效率问题,用软件模拟的 CPU 比硬件直接实现的 CPU 慢很多。如在 MIPS 计算机上使用二进制翻译的方法运行 ARM 程序,比起把该程序直接从源代码编译成 MIPS 指令并在 MIPS 计算机上执行,运行速度一般有数量级的差异。

本文针对安卓应用程序在 MIPS 平台上不兼容的原因,设计了二进制翻译系统来解决安卓应用程序在 MIPS 平台上的不兼容问题。设计的二进制翻译系统实现了动态库的跨平台加载,并采用动态二进制翻译技术,将 ARM 动态库中的二进制指令翻译成为 MIPS 架构的二进制指令,从而解决安卓应用的兼容性问题。

1 相关工作

libhoudini[9]是由美国 Intel 公司开发的项目,用于在基于 X86 处理器平台的安卓系统上支持 ARM 的二进制运行,解决安卓应用程序的兼容性问题。libhoudini 把 ARM 的二进制代码动态翻译成 X86 处理器平台上的可执行代码。

Qemu[10]是由 Fabrice Bellard 开发的开源的纯软件虚拟化模拟器,几乎可以模拟任何硬件设备。Qemu 主要是提取源体系结构代码,然后将其翻译成可信计算组织(trusted computing group, TCG)中间代码,然后将 TCG 中间代码翻译成目标体系结构代码。它支持 X86、ARM、MIPS、SPARC 和 Power PC 等架构。

DigitalBridge 是中国科学院计算所开发的一个固定源和固定目标的二进制翻译器[11-12],能将 X86 程序运行在 MIPS 平台上,支持 Adobe Reader、Firefox、MySQL、Apache 等应用程序。DigitalBridge 在动态翻译过程中,先不计算标志寄存器的结果,使用延迟技术,然后在下一阶段收集程序运行的信息。根据这一信息对前一阶段生成的代码进行在热路径的优化,这个优化主要是在热路径上进行。这样系统可以首先迅速生成执行效率比较差的代码。在实际的优化过程中,要针对标志位计算进行优化,采用两种方式,针对基于模式的指令组合翻译优化和针对延迟标志位计算的优化。DigitalBridge 虽然使用两种模式处理指令流,但仍需对只执行一次的代码生成基本块,生成基本块的操作开销比解释执行大[13]。

2 MIPS 安卓平台上 ARM 二进制翻译系统的设计

2.1 动态库跨平台二进制兼容模型

安卓应用的开发工具包括安卓 SDK 和 NDK。其中,SDK 是安卓软件开发工具包,使用该工具包开发的应用程序采用 Java 语言,不存在兼容性问题。NDK 是安卓本地开发工具包。采用 NDK 开发的应用程序可能调用了本地库,本地库代码是和某种架构对应的机器码。如果安装平台的架构和应用程序中调用的动态库不匹配,则无法正常执行,即出现不兼容。

由上述分析可知,造成 MIPS 平台上安卓应用程序不兼容的原因是使用了 JNI 函数的安卓应用程序中没有 MIPS 版本的动态库,导致程序运行过程中,加载动态共享库时出错。针对这一问题,本文提出了一种动态库跨平台二进制兼容模型,并根据该模型提出通用的翻译系统架构。

动态库跨平台二进制兼容模型所要解决的问题是要通过二进制兼容技术,使得动态链接库中的二进制代码段能够在不同体系结构的硬件平台上被正确加载、链接及执行。携带第三方动态库的程序由目标体系结构的主程序和源体系结构的动态库两部分组成,图 1 所示为该模型的结构图。该模型分为动态链接和代码执行阶段。在动态链接阶段,目标体系结构的主程序调用源体系结构的共享库时,通过定制的链接器将源体系结构的共享库进行动态链接加载;在程序执行阶段,目标体系结构的二进制代码直接运行在 CPU 上,而源体系结构的共享库代码

通过二进制翻译技术在目标体系结构 CPU 上运行。

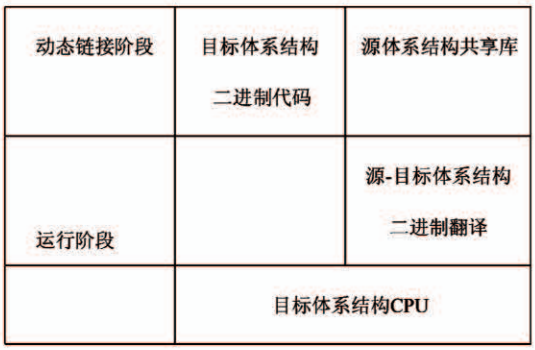

图 1　动态库跨平台二进制兼容模型

根据动态库跨平台二进制兼容模型,本文提出通用的跨平台二进制代码翻译系统架构(如图 2 所示),主要包含 3 个功能模块。

(1) 自定义动态链接器。由于目标体系结构平台上的动态链接器在加载源体系结构平台动态库时会出错,故要完成第三方动态库在目标体系结构平台上的加载,就需要提供一个工作在目标体系结构平台上的自定义动态链接器完成工作。主要完成符号决议、重定位等链接工作。

(2) 二进制翻译器。在完成源体系结构平台动态库在目标机上的动态加载链接工作后,根据调用函数的名称得到跳转地址,但此时跳转到的二进制代码是不同体系结构的代码,因此需要一个二进制翻译器模块将源体系结构平台二进制指令翻译为目标体系结构平台的二进制指令。

(3) 控制核心。在指令的执行过程中,需要对指令处理的中间结果进行存储,并且维护虚拟 CPU 的寄存器状态。另外,当所调用的函数为系统调用时,需要将该系统调用转换成本地系统调用。

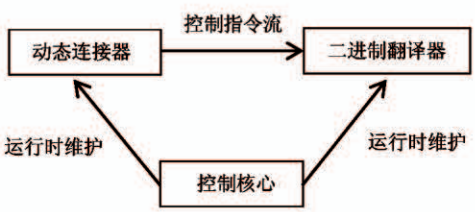

图 2　跨平台二进制代码翻译系统架构

本文二进制翻译系统的目标体系结构平台是 MIPS,而程序所携带的动态库是源体系结构 ARM 的动态库文件。在图 2 所示的跨平台二进制代码动态翻译系统架构中,动态链接器主要用于在 MIPS 二进制代码中链接 ARM 共享库文件,二进制翻译器主要用于将 ARM 二进制代码翻译为 MIPS 代码来执行。下面具体介绍二进制翻译系统各模块的设计。

2.2　二进制翻译器的设计

二进制翻译器设计的依据是目标机器的指令集架构,不同体系结构的指令集在指令架构、寻址方式、内存访问形式上都有很大的不同。设计一个二进制翻译器需要主要包含 2 个基本功能。

(1) 解释源平台二进制指令。根据源平台指令的格式对二进制指令进行解析,分析出指令的操作码、操作数、执行条件等信息。

(2) 在目标平台上执行源指令等价操作。根据获得的指令信息将源指令转换成在目标平台上能够执行的微操作或者目标代码,并执行生成的目标代码实现源指令的等价操作。

本系统的动态二进制翻译模块整体结构如图 3 所示,由系统控制核心、ARM 指令译码器、微码转换器、编译器 4 部分组成。其中控制核心、ARM 指令译码器、微码转换器是系统的运行时模块,编译器是系统编译时依赖的编译工具,负责将解释器中的 C 微码指令编译成目标平台的二进制代码。

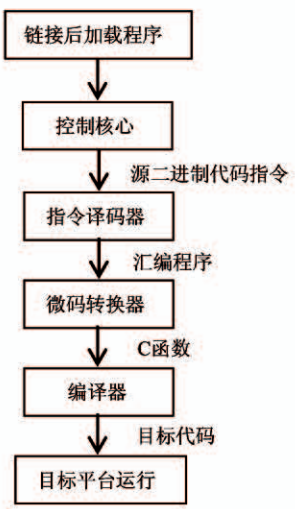

图 3　二进制翻译器的流程图

二进制翻译模块运行时,源平台的所有资源都通过 CPU 基址加上特定的偏移量来访问。系统控制核心首先会根据动态链接器对 ARM 动态库的加

载和分析，返回文件中所调动的子函数的入口位置，从而获得文件执行的代码段入口地址，并存放在程序计数器中。然后控制核心读取一条 ARM 二进制指令并传入 ARM 指令译码器，译码器按照 ARM 指令的格式和译码规则翻译为汇编指令，并将形成的汇编指令送入微码转换器。微码转换器会将接收到的汇编指令翻译成等价的 C 函数微码片段。运行时通过动态代码生成器把以上微操作组合成一个函数并执行，就相当于执行了一条源指令，通过执行这些指令流完成对源平台指令的等价执行。

2.3　控制核心的设计

ARM 二进制代码动态翻译系统需要一个控制核心来在运行时记录环境变量，该控制核心需要包含以下 3 个基本功能。

（1）系统初始化。系统初始化主要包括虚拟 CPU 寄存器的初始化，代码区和栈区的内存空间的分配以及其他系统参数的初始化。系统初始化过程中，由于不同体系结构下寄存器的名称、数量、位数及功能不相同，需要根据具体目标平台设置相应的系统初始化流程。

（2）运行时管理。运行时管理是对运行时的内存数据和目标机的虚拟 CPU 寄存器状态的管理。运行时管理模块包含线程共享的堆数据管理、线程独享的栈数据管理以及程序计数器的管理等，具体实现可根据虚拟内存的布局进行具体的管理。运行时的虚拟 CPU 管理主要是根据程序的执行情况更新虚拟 CPU 的寄存器状态，包括通用寄存器、段寄存器、标志位寄存器等。此外，运行时管理的一个重要工作是处理两种不同的架构之间代码相互调用，为此准备必要的胶合代码。

（3）系统调用的处理。劫持系统调用，并将源平台的系统调用转换成本地系统调用。常用的本地系统调用劫持的方法有：1）通过修改 sys _ call _ table 中对应的内核函数地址进行系统调用劫持。2）对于特定的系统调用，通过修改 vfs 文件系统进行劫持。3）构造新的中断描述符替换原系统调用使用的中断号，并在新的中断处理函数中处理系统调用。但是，第三方动态库中的系统调用劫持和本地系统调用劫持不一样，实际上是将源平台系统调用转换到本地平台，比较简单的方法就是在进行第三方动态库中的系统调用过程中，将系统调用相关函数转化为本地系统调用，在调用时获取翻译系统此时的寄存器和内存状态，转化为 MIPS 进行系统调用的寄存器和内存状态。接着在 MIPS 中调用本地的系统调用，获取结果，并返回给翻译系统。

2.4　动态链接器设计

动态链接器的流程图如图 4 所示。动态链接器包含链接器和加载器[14]，但在实际的 Linux 系统中，任何一个链接器需要实现以下 3 个基本功能。

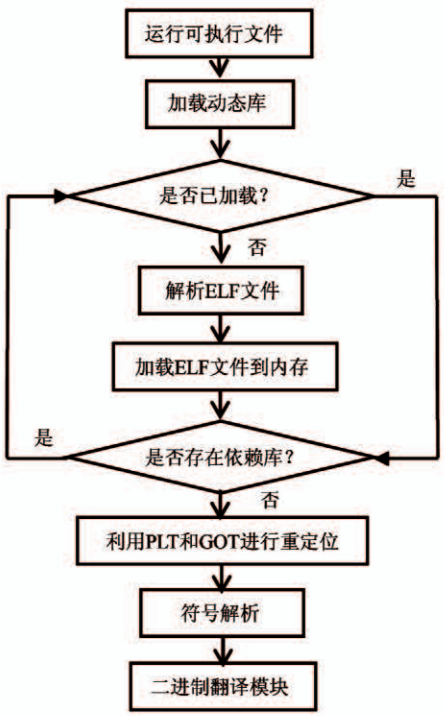

图 4　动态链接器的流程图

（1）程序加载。根据动态库的名称将程序从磁盘拷贝到主内存中，需要注意的是当程序加载异架构的动态库文件失败时，则用自己开发的二进制模块实现对动态库的装入和链接。共享库文件在内存加载的过程中根据 ELF 文件[15]的段表对共享库文件的 ELF 头部（ELF Header）、PLT 表、GOT 表、字符串表（String Table）以及其他链接所需要的所有节区进行解析，同时记录过程中所需要的属性值。

（2）重定位。根据加载地址调整程序中的数据和代码以使程序的变量和函数地址指向实际逻辑地址。需要注意的是在对第三方动态库的重定位过程

中对系统调用的处理。系统调用是用户程序与内核间的接口,内核函数可以驱动硬件设备,因为不同硬件设备的内核函数实现不同,所以需要将原系统调用进行捕获转移到目标机的系统调用上。

(3) 符号解析。可执行文件和动态库文件间的变量和函数的相互引用是通过符号进行的,需要链接器解析符号来给引用符号分配在库中的地址,使得 call 指令时直接调用该地址。在共享库的符号解析过程中,实际是解析. dynsym 节区中保存的动态符号表的过程。

3 ARM 二进制翻译系统的实现

3.1 二进制翻译模块

二进制翻译模块主要负责第三方动态库的二进制代码翻译工作,基于二进制翻译技术的分析对比,本系统中二进制翻译模块基于开源项目 Unicorn[16] 进行开发,Unicorn 基于动态翻译技术。使用 Unicorn 引擎的基本流程是:(1)初始化二进制翻译器,初始化一个 64 bit 的 ARM 架构的模拟器;(2)映射二进制翻译器的虚拟地址,将代码区映射到 0x10000、栈区映射到 0x20000 的内存空间;(3)把二进制数据写入二进制翻译器的虚拟地址;(4)初始化二进制翻译器的寄存器;(5)设置 hook 回调函数,其中 hook _ code 就是一个回调函数,可以用来打印一些中间执行结果,由用户自己实现。Unicorn 的 hook 回调机制极大地方便了对指令的调试,通过 hook 机制查看当前的寄存器和栈区的数据;(6)执行指令。

在指令译码器设计方面,指令译码器所要执行的任务是将一条二进制指令翻译成其对应的汇编指令,也就是反汇编的过程。当获得一条二进制指令时,根据 ARM 指令集的指令格式,获取执行指令的操作码、操作数、条件码等信息。这一步骤的重点在于获取不同寻址方式下的操作数,通过分析指令字助记符和指令码,可以很方便地得到不同寻址方式的操作数。

在微码转换器设计方面,微码转换器根据指令译码器产生的指令操作码、操作数个数、寻址模式等信息,将汇编指令转换成对应功能的 C 语言中间代码,采用类似 Qemu 的微操作形式。例如,当遇到如下 ARM 指令:

```
Add r3 r3 16 # r3 = r3 + 16
```

微码转换器将其分解为如下的微操作。

```
movl _ T0 _ r1 # T0 = r1
addl _ T0 _ im 16 # T0 = T0 +16
movl _ r1 _ T0 # r1 = T0
```

随后在转换器中会找出这 3 条指令的对应 C 函数为

```
void op _ movl _ T0 _ r1(void){
    T0 = env- > regs[1];}
void op _ addl _ T0 _ im(void){
    T0 = T0 + ((long)(& _ op _ param1));}
void op _ movl _ r1 _ T0 (void){
    env- > regs[1] = T0;}
```

而立即数 16 会被送入控制核心中保存立即数的堆栈中以备运行时使用。本系统对中间代码转换函数进行了简化,直接将 C 函数中的代码内联到微码转换环节,"Add r3 r3 16"指令将被直接转换为如下微码。这些微码片段在编译二进制翻译模块源代码的时候就已经被编译成了 MIPS 平台的二进制代码。

```
T0 = env- > regs[1];
T0 = T0 + ((long)(& _ op _ param1));
env- > regs[1] = T0;
```

3.2 控制核心模块

控制核心中需要实现包括不同架构的函数互调用、处理系统调用、地址空间协调等功能。

不同架构有不同的应用程序二进制接口(application binary interface,ABI)。ABI 是在二进制级别上定义的一套函数调用规范,涵盖诸如数据类型、大小、对齐方式、参数如何传递并返回值等。控制核心模块必须处理 ARM ABI 和 MIPS ABI 之间的差异。针对 MIPS 代码调用 ARM 本地库的情况,在龙芯安卓系统中,调用本地库是通过 dvmPlatformInvoke 实现的。dvmPlatformInvoke 的功能是根据调用函数的签名来为该函数准备调用环境,从 Java 的运行数据里取出函数的参数,按 C 程序的要求准备好寄存器

和调用栈,然后调用本地库的函数。对函数 func,该函数定义为 int func(int *arg*1, int *arg*2)就是要取出两个整型的参数,放到 a0/a1 两个寄存器里边。如果 func 的实现是 MIPS 代码,它会正确地按照 MIPS 的 ABI 约定,从 a0/a1 寄存器取出参数。但是在目标是 ARM 代码的情况下,本地库里的代码只会按 ARM 的 ABI 约定,试图从 ARM 架构的 r0/r1 寄存器得到参数。所以,控制核心模块必须处理两者之间的差异,先按 MIPS 的 ABI 取出参数,再按 ARM 的 ABI 准备参数和调用环境。栈和返回值也需要根据情况处理。另外,控制核心还要处理从 ARM 代码库调用 MIPS 代码等情况。

在系统调用方面,系统调用指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务。但不同体系结构内核函数的实现并不相同,需要将原系统调用进行捕获转移到目标机的系统调用上。系统调用实际使用时通常会被包装成各种库函数,可以直接劫持库函数调用,改为使用主机的相应函数。涉及的库函数中绝大部分都是 linux 系统中最底层的 api—glibc 中的库函数,包括数据转换函数、字符串处理函数、时间函数、I/O 函数、格式化输入输出函数、文件及目录函数、Socket 相关函数、消息队列函数、pthread 相关函数等。基于这个原理,在系统中定义一个 plt _ stubs 表用于记录 glibc 中的库函数名,并对每个库函数编写对应的转换函数。另外,对于一些特殊的库函数,除参数传递外,还需要做一些本地化工作,例如涉及内存申请的 malloc 动态内存分配函数。

在地址空间协调方面,主机函数和 ARM 本地库互相调用会造成一个问题。ARM 本地库是由一个虚拟 CPU 执行,通常它也有自己的地址空间,因此在 ARM 函数看到的地址和主机函数看到的地址可能不一样。这样会带来问题,如从参数传递过来的数据结构里边如果有地址都需要转换。为了解决这个问题,本文利用 unicorn 的地址空间映射接口,统一了 ARM 虚拟机和主机进程的地址空间,让两边看到的地址一致。

3.3 与 Android 系统的整合

需要修改虚拟机代码,将二进制翻译框架集成到 Android 系统中,主要修改如下。

(1) 装入和链接。如果打开 mips 的本地库失败,则尝试打开 arm 的本地库,并采用所开发的二进制模块提供的查找符号以及调用函数等接口。

(2) 调用。在 dvmPlatformInvoke 做一些处理。仅用 method- > insns 传递函数指针不好区分是本地库函数还是 arm 库函数,需要传递整个 method 指针,并且在之前的查找 method 时记录下所在的 ShareLib 结构指针。这样,对于 arm 库函数,可以从 method 结构查找出对应的 ShareLib,然后用它找到二进制翻译模块的 vm 指针和其中的库文件句柄,并做好二进制翻译需要的调用数据准备,再进行函数调用。

4 实验平台和实验结果

4.1 实验平台

实验平台为基于 MIPS 架构的龙芯处理器平台(配置如表 1 所示),采用的 CPU 为四核龙芯 3A3000,基于 GS464E 处理器核架构设计[17],主频为 1.5 GHz,片上三级 Cache 为 4 MB;采用的操作系统为龙芯自研的 Loongnix 1.0,其中内核为 Linux 3.10.0,内存容量为 8 GB。

表 1 测试主机的配置

CPU 型号	龙芯 3A3000
处理器核数	4 核
主频	1.5 GHz
OS	Loongnix 1.0
Kernel	Linux 3.10.0
内存	8 GB

4.2 实验结果

安卓 ARM 动态库兼容系统测试包含功能测试和性能测试两个部分,功能测试主要验证安卓应用在 MIPS 平台上运行的有效性、基本的算数逻辑运算、常用的库函数调用的正确性;性能测试通过对比使用二进制翻译器的执行效率与采用本地编译执行效率的差异,从而提出未来优化建议。

4.2.1 功能测试

(1) 测试集测试

测试集测试的目的是测试基本的算数逻辑运算、常用的库函数调用的正确性。所编写的 C 语言测试程序,通过安卓 NDK 内的 gcc 编译成 ARM 动态库文件,最后使用本文的二进制翻译模块执行该动态库中相应的测试函数来记录运行结果。

Basic 测试集中包含 for 循环体、赋值操作、算术和逻辑运算,编译之后的二进制文件能覆盖基本的 ARM 指令,运行后能验证指令的正确性。其他的测试集用于测试系统调用处理的正确性,安卓应用程序中,常用的系统调用是通过调用 glibc 库中的库函数实现的,常用的库函数有 I/O 函数、格式化输入输出函数、套接字函数、线程函数、时间函数、字符串处理函数、文件读写函数、数据转换函数、文件及目录函数。针对这些常用的库函数,分别设计一个测试集用于测试相关功能是否正常运行,详细信息见表 2。测试集功能测试结果如表 3 所示。

表 2　测试集设计

序号	测试集	测试项内容
1	Basic	基本算术与逻辑运算
2	Output	IO、格式化输入输出函数
3	Socket	套接字库函数
4	Pthread	线程相关库函数
5	Time	时间相关库函数
6	String	字符串相关库函数
7	Mq	消息队列相关库函数
8	Data	数据转换相关函数
9	File	文件读写相关库函数

表 3　测试集设计

序号	测试集	测试结果
1	Basic	正确
2	Output	正确
3	Socket	正确
4	Pthread	正确
5	Time	正确
6	String	正确
7	Mq	正确
8	Data	正确
9	File	正确

(2)应用测试

应用测试的目的是测试二进制翻译器的实际兼容效果,测试程序为应用市场上下载的 WPS、美图秀秀、福昕 PDF、网易云音乐等典型应用,应用能够正常打开,使用各种基本功能没有出现 bug,应用测试正常。

测试结果说明二进制翻译能正常进行基本的运算,并能够对系统调用进行正常处理。

4.2.2　性能测试

本文采用常用于嵌入式系统性能测试的 EEMBC-Coremark 基准测试程序[18] 和包含 30 个数值计算小程序的 Polybench/C 4.2[19] 对本文二进制翻译的性能进行了测试。为了测试方便,为本文的二进制翻译模块添加了直接执行二进制文件的功能。

Coremark 用缺省参数编译,取性能运行结果。Coremark 性能测试结果如表 4所示。

表 4　Coremark 性能测试结果

测试项	本地运行分数	二进制翻译运行分数	比例
Coremark	2776	752	27.0%

Polybench 4.2 全部采用默认的数据集,加-DPOLYBENCH _ TIME 参数编译以打印出执行时间。性能测试结果如表 5 所示。

从测试结果可以看出,对于以定点为主的 Coremark 基准测试程序,本文的二进制翻译模块运行速度能够达到本地编译运行速度的 27%。然而对于以数值计算为主的 Polybench,该比例平均只有 4.1%。总体来说,安卓 ARM 动态库兼容系统在功能上正确翻译执行了第三方动态库中的函数代码,但在性能上相对源码编译执行有较大下降,尤其是浮点程序。这符合 Qemu 本身二进制翻译引擎的特点(浮点模拟效率较低)。同时 Unicorn 对 Qemu 有修改,这些修改可能会对二进制翻译器的性能有所损耗。未来工作需要进一步优化其性能。

5　结　论

本文提出了一种动态库跨平台二进制兼容模

表 5 Polybench 性能测试结果

测试项	本地运行时间/s	二进制翻译运行时间/s	比例/%
correlation	9.14	262.04	3.5
covariance	9.14	260.99	3.5
2mm	12.32	495.30	2.5
3mm	20.28	701.97	2.9
Atax	0.03	2	1.5
Bicg	0.11	1.98	5.6
doitgen	3.26	132.42	2.5
Mvt	0.06	2.06	2.9
Gemm	4.47	467.37	1.0
gemver	0.08	4.74	1.7
gesummv	0.05	0.83	6.0
symm	14.73	349.99	4.2
syr2k	13.50	500.29	2.7
syrk	3.71	245.43	1.5
trmm	4.05	153.49	2.6
cholesky	8.29	334.66	2.5
durbin	0.02	1.02	2.0
gramschmidt	12.04	389.80	3.1
lu	20.44	702.84	2.9
ludcmp	19.68	634.93	3.1
trisolv	0.01	0.50	2.0
deriche	2.11	18.40	11.5
floyd-warshall	148.41	842.14	17.6
nussinov	17.86	145.37	12.3
adi	112.78	1501.55	7.5
fdtd-2d	13.74	776.99	1.8
head-3d	26.88	2257.78	1.2
jacobi-1d	0.01	0.69	1.4
jacobi-2d	12.85	914.88	1.4
seidel-2d	168.12	1896.02	8.9
平均			4.1

型,以及通用的跨平台二进制翻译系统的架构,并在 MIPS 安卓平台上设计实现了 ARM 二进制翻译系统。该系统能够进行 ARM 动态库的跨平台加载,并采用动态二进制翻译,将 ARM 动态库中的二进制指令翻译成为 MIPS 架构的二进制指令,从而使 ARM 二进制翻译系统能够在 MIPS 架构处理器上执行。

应用测试结果表明,该系统可以运行调用 ARM 本地库的安卓应用程序,解决了 MIPS 平台安卓应用的兼容性问题。采用定点为主的 EEMBC-Coremark 测试结果表明,二进制翻译执行获得了本地编译执行 27% 的性能,在现在 CPU 运算能力很强的情况下,应用性能能够接受。对于数值计算为主的 Polybench 而言,二进制翻译执行性能与本地编译执行的性能差距较大,未来工作主要是进一步提升 ARM 二进制翻译系统的性能。

参考文献

[1] 胡伟武,唐志敏. 龙芯 1 号处理器结构设计[J]. 计算机学报,2003,26(4):2-13

[2] Hu W W, Zhang F X, Li Z S. Microarchitecture of the Godson-2 processor[J]. *Journal of Computer Science and Technology*, 2005, 20(2):243-249

[3] Hu W W, Wang J, Gao X, et al. Godson-3: a scalable multicore RISC processor with x86 emulation[J]. *IEEE Micro*, 2009, 29(2):17-29

[4] 王向辉,张国印,等. Android 应用程序开发[M]. 北京:清华大学出版社,2010

[5] Gordon R. Essential JNI: Java Native Interface[M]. London: Prentice Hall, 1998

[6] Altman E R, Kaeli D, Sheffer Y. Welcome to the opportunities of binary translation[J]. *Computer*, 2000, 33(3):40-45

[7] Cifuentes C, Malhotra V M. Binary translation: static, dynamic, retargetable? [C]//Proceedings of IEEE International Conference on Software Maintenance, Washington DC, USA, 1996:340-349

[8] Zheng C, Thompson C. PA-RISC to IA-64: transparent execution, no recompilation[J]. *Computer*, 2000, 33(3): 47-52

[9] Android x86 介绍[EB/OL]. https://www.android-X86.org: Android-x86, 2019

[10] Bellard F. Qemu, a fast and portable dynamic translator [C]//Proceedings of Usenix Technical Conference, Vancouver, Canada, 2005:41-41

[11] 谢海斌, 武成岗, 张兆庆, 等. 动态二进制翻译中的代码 Cache 管理策略[J]. 计算机工程, 2005, 31(10):97-99

[12] 王文文,武成岗,白童心,等. 二进制翻译中标志位的模式化翻译方法[J]. 计算机研究与发展,2014,51(10):2336-2347

[13] 唐锋. 动态二进制翻译优化研究[D]. 北京:中国科学院大学, 2006

[14] Levine J R. 链接器和加载器[M]. 李勇,译, 北京: 北京航空航天大学出版社, 2009

[15] 何先波, 唐宁九, 吕方,等. ELF 文件格式及应用[J]. 计算机应用研究, 2001,18(11):144-145

[16] Unicorn. Next generation CPU emulator framework[EB/OL]. http://www.unicorn-engine.org/BHUSA2015-unicorn.pdf: Unicorn, 2015

[17] 吴瑞阳,汪文祥,王焕东,等. 龙芯 GS464E 处理器核架构设计[J]. 中国科学:信息科学,2015,45(4):480-500

[18] EEMBC-Coremark[EB/OL]. https://www.eembc.org/coremark/scores.php:EEMBC, 2019

[19] Polybench/C 4.2[EB/OL]. http://sourceforge.net/projects/polybench/files/lastest/download: Slashdot Media, 2019

ARM binary translation system based on MIPS Android platform

Zhao Baohua[*,**,***], An Ningyu[**,***], Xu Zhechong[****], Du Anli[****], Su Tao[****]

([*]Faculty of Information Technology, Beijing University of Technology, Beijing 100124)

([**]Global Energy Interconnection Research Institute Co. Ltd, Beijing 102209)

([***]Artificial Intelligence on Electric Power System State Grid Corporation Joint Laboratory (GEIRI), Beijing 102209)

([****]Loongson Technology Co. Ltd, Beijing 100095)

Abstract

Microprocessor without interlocked piped stage (MIPS) is an important processor architecture. Android is the mainstream operating system of mobile terminals. Running Android operating system on MIPS architecture processor can effectively expand the field of application, but the problem is that the Android application that calls the local library of advanced RISC machine (ARM) architecture can not run, that is, there is application incompatibility problem. This paper proposes a dynamic library cross-platform binary compatibility model, designs and implements an ARM binary translation system based on MIPS Android platform, which can load dynamic libraries across platforms and translates the binary instructions in ARM dynamic libraries into MIPS binary instructions, so that they can be executed on MIPS architecture processors. The experimental results show that the system can run Android applications calling ARM local library and solve the compatibility problem of Android applications.

Key words: binary translation, advanced RISC machine (ARM), microprocessor without interlocked piped stage (MIPS), Android