文章编号:1674-2974(2024)08-0070-08 DOI:10.16339/j.cnki.hdxbzkb.2024279

基于QEMU的SIMD指令替换浮点指令框架

刘登峰,李东亚,柴志雷,周浩杰†,丁海峰
(江南大学 人工智能与计算机学院,江苏 无锡 214122)

摘 要:现在,几乎每个处理器架构都已经加入了对SIMD(single instruction multiple data)指令的支持,SIMD指令能同时对一组数据执行相同的操作,通过数据并行来提高处理器的处理性能.但是大部分动态二进制翻译器忽略了本地SIMD指令的利用,而是以软件语言实现来模拟浮点计算.本文提出了一种基于QEMU翻译系统的FP-QEMU框架,FP-QEMU框架采用SIMD指令来优化替换浮点计算指令,并在X86和ARM平台上完成了完整的浮点实现.该框架可以识别动态二进制翻译系统中的浮点计算优化机会并利用SIMD指令来提升系统翻译的性能.采用SPEC 2006作为测试基准,实验表明相比QEMU,FP-QEMU跨平台的ARM应用在X86计算机上运行的最高加速比可达51.5%,平均加速比达到37.42%.

关键词:SIMD;QEMU;动态二进制翻译;浮点计算
中图分类号:TP314 **文献标志码**:A

QEMU-based Framework for SIMD Instruction Replacement Floating-point Instructions

LIU Dengfeng, LI Dongya, CHAI Zhilei, ZHOU Haojie†, DING Haifeng
(School of Artificial Intelligence and Computing, Jiangnan University, Wuxi 214122, China)

Abstract: Now, almost every processor architecture has added support for SIMD (single instruction multiple data) instructions. SIMD instructions can perform the same operation on a set of data simultaneously, enhancing the processing performance of the processor through data parallelism. However, most dynamic binary translators ignore the use of native SIMD instructions and instead simulate floating-point computations in software languages. This paper proposes a framework called FP-QEMU, based on QEMU translation system. FP-QEMU adopts SIMD instructions to optimize and replace floating-point calculation instructions, and completes a complete floating-point implementation on X86 and ARM benchmark platforms. The framework can identify the optimization opportunities of floating-point computation acceleration in dynamic binary translation system and use SIMD instructions to achieve the effect of improving the translation performance of dynamic binary translation system. Using SPEC 2006 as the benchmark, experiments show that compared with QEMU, FP-QEMU cross-platform ARM applications running on X86 computers can achieve a maximum speedup of 51.5% and an average speedup of 37.42%.

* **收稿日期**:2023-11-30
基金项目:国家重点研发专项计划项目(2022YFE0112400),National Key R&D Program of China(2022YFE0112400);国家自然科学基金资助项目(21706096),National Natural Science Foundation of China(21706096);江苏省自然科学基金青年项目(BK20160162),Youth Project of Natural Science Foundation of Jiangsu Province(BK20160162)
作者简介:刘登峰(1980—),女,河南南阳人,江南大学副教授,博士
† 通信联系人,E-mail: zhouhaojie@jiangnan.edu.cn

Key words:SIMD;QEMU; dynamic binary translation ;floating-point arithmetic

动态二进制翻译(dynamic binary translation)技术作为一种特殊的即时编译技术,能够将在目标平台(Target)上编译的二进制文件运行在其他的体系架构主机平台(Host)上,而不要求被翻译程序的源代码作为翻译系统的输入,降低了应用程序和底层硬件之间的耦合度,为新旧处理器互相兼容以及跨平台软件迁移架起了一座桥梁,因而在体系结构优化、程序性能优化、安全分析以及软件移植的研究中备受关注.

动态二进制翻译对用户可以做到完全透明,无需用户干预,但是动态二进制翻译的整体效率并不高[1].以QEMU为例,目前QEMU[2]在实现对多目标机和多宿主机兼容的过程中,并未充分利用宿主机体系结构资源与优势,未经优化的二进制翻译器翻译效率通常只有本地编译执行的10%左右[3].为了提高动态二进制翻译的效率,深入本地平台(Host)的体系架构特征,充分利用本地平台的资源优势成为相关研究的热点.

针对上述情况,本文提出了FP-QEMU框架,使用本地SIMD指令替换二进制翻译中的浮点计算指令来达到提升整体效率的目标.FP-QEMU对源程序中浮点处理函数的执行进行了本地SIMD指令的替换处理,以充分发挥本地平台的架构优势,与处理浮点计算指令时纯软件运算中指令逐条运算的方法相比较,当需要对一组数据进行相同的运算操作时,使用SIMD指令能显著提高处理性能.本文以QEMU作为基础验证平台进行了实验验证,实验数据表明,相比于QEMU,FP-QEMU平均效率提升了37.42%.

1　动态二进制翻译

1.1　QEMU

QEMU是当前主流的多平台开源二进制翻译框架,具有良好的可移植性和可扩展性.基于QEMU近年来涌现出了大量研究工作,包括寄存器分配优化[4]、向量优化[4]、中间代码优化[5]、基本块链接优化[6]和多线程优化[6]等,同时衍生出大量工具链,包括基于LLVM优化加速的多线程翻译器HQEMU[6]、缓存独占的并行全系统模拟器COREMU[7]、缓存共享的并行全系统模拟器PQEMU[8]、符号执行与二进制翻译融合的SymQEMU[9]、基于分布式框架的DQEMU[10]、兼容Pin[11]的二进制插桩工具PEMU[12]、基于多面体优化对程序并行化的LLPEMU[13]和动态二进制分析工具PANDA[14]等.

QEMU整体工作流程如图1所示.

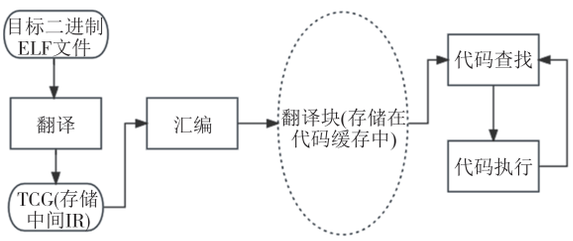

图1　QEMU工作流程

Fig.1　QEMU workflow

1.2　浮点计算优化

浮点数是计算机近似地表示任意某个实数,这种表示方法类似于基数为10的科学计数法.浮点计算是指浮点数参与的运算,这种运算通常伴随着因为无法精确表示而进行的近似或舍入.与相同位数的整数相比,浮点数的表示范围更广,精度更高.SPEC2006的基准测试套件包含SPEC整型基准和SPEC浮点型基准,其中SPEC浮点型基准中涵盖了多种密集型浮点计算任务,可以通过测试SPEC2006中浮点型课题来评估计算机系统在处理浮点型计算任务时的性能,进行性能比较、系统优化和应用优化,以满足在不同应用场景下对比FP-QEMU和QEMU在浮点计算性能方面表现效果的需求.

在二进制翻译过程中,浮点计算优化是一项重要的工作,许多研究团队和学者针对浮点计算优化已经开展了一系列的研究.文献[1]提出的针对QEMU的浮点库函数本地化的方法对于库函数处理的加速类似于并行加速,间接加速了浮点计算的过程.文献[4]提出的针对QEMU的浮点处理函数简化方法在Nbench测试集上具有平均加速比11.92%的加速效果.文献[4]提出的宿主机平台与本地平台的浮点寄存器直接映射方法在Nbench测试集上也取得成果,代码量平均减少30.91%.

以上介绍的几种浮点计算优化方法都是从单个技术点作为切入点,并不具有通用性,没有从系统框架角度出发,不具有普适性.因此,结合上述情况,本文提出了系统化SIMD指令替换浮点计算指令框架FP-QEMU,通过采用SIMD指令替换浮点计算指令

的方法来充分发挥本地平台的资源与优势,同时适用于各种支持SIMD指令的体系架构平台,此外本文提出的FP-QEMU框架具有高拓展性,为后续研究人员提供了良好基础.

1.3 SIMD指令

SIMD指令是一种处理器指令集架构,SIMD指令允许处理器同时对一组数据进行相同的操作,以达到高效的并行计算.通过在单个时钟周期内对多个数据元素执行相同的操作,SIMD指令可以提高数据并行性,减少指令执行的开销.SIMD指令与浮点运算密切相关,通过使用SIMD指令,处理器可以在一条指令中同时对多个浮点数进行运算.例如,可以一次性对一个向量或标量中的多个浮点数进行加法、乘法、平方根等操作,从而减少指令的数量和执行的开销,进而提高浮点运算的效率和性能.

目前,许多软件都采用SIMD指令技术进行自身优化,但是二进制翻译系统对于SIMD指令的处理仍有不足,没有充分利用新型体系结构中SIMD技术的优势.因此翻译性能较低.

科研人员针对上述情况进行了大量研究实验,在研究过程中提出了若干高效利用本地SIMD指令的方法.文献[15]提出利用GCC向量拓展来使后端生成SIMD指令,并在X86-64平台的HQEMU中实现了相关功能,但没有兼容其他平台.文献[16]提出了一种SIMD数据类型跟踪算法跟踪SIMD寄存器中数据元素的类型,其专注于SIMD寄存器映射而不是SIMD代码转换,可以减少数据移动的数量,并在X86平台实现.尽管在学术界存在许多专注于浮点计算和SIMD指令研究的学者,然而对于同时兼容多个平台且具有高普适性的研究而言,研究成果相对较少,这表明该领域具备广阔的研究空间和潜力.

2 FP-QEMU框架

2.1 FP-QEMU框架原理图

综合前文所述,本文提出了一种解决动态二进制翻译低下问题的方法.本文通过采用SIMD指令替换浮点计算指令方法,充分利用本地平台的SIMD资源实现二进制翻译应用加速.

结合前文分析,SIMD指令可以同时对多个数据元素执行相同的操作,以SSE指令为例,SSE指令可以执行诸如加、减、乘、除等浮点数运算.此时SSE指令可进行替换浮点计算指令的操作,基本操作原理是使用向量化计算的方式,将多个浮点数数据打包成一个向量(例如128位或256位),然后通过一条SSE指令对整个向量进行并行操作.这样,就能够在单个指令周期内同时完成多个浮点数的计算,从而提高计算效率.总的来说,SIMD指令替换浮点计算指令是通过向量化计算,将多个浮点数数据打包成一个向量,然后通过一条SIMD指令对整个向量进行并行操作,进而实现提高计算效率的目标.

FP-QEMU核心在于动态二进制翻译过程中实时检测分析翻译过程中的操作数和操作码,并且对不同的操作数和操作码进行具体分类.FP-QEMU提取操作数并进行分析判断操作数的精度范围,同时分析操作码中具体运算类型.FP-QEMU总结得出以下分类.分类1:浮点运算中操作码是加、减、乘、除等运算且操作数精度为双精度(64位)或单精度(32位).分类2:浮点运算中操作数精度超过64位精度或操作码中运算属于复杂运算,如开方和三角函数以及工程和金融上的应用等.针对分类1本文采用SIMD指令使用数据并行处理的方法来替换此类运算,分类2仍然使用浮点计算单元进行计算.

FP-QEMU框架整体结构图如图2如示.

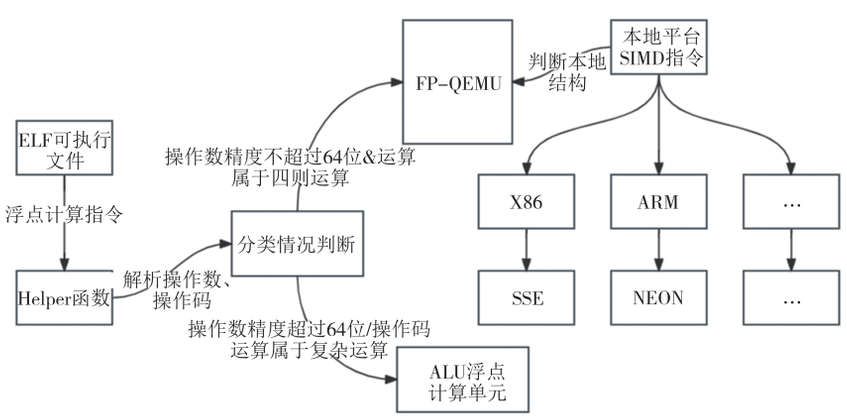

图2 FP-QEMU框架整体结构图

Fig.2 Overall structure diagram of FP-QEMU framework

本文采用状态转移流数学模型来表示FP-QEMU框架中SIMD指令替换浮点指令策略的工作流,FP-QEMU的状态转移图如图3所示.

2.1.1　状态定义

a)ELF: target架构的可执行文件.

b)Helper: 运算属于浮点计算范畴时系统调用Helper函数处理运算.

c)SIMD Exchange: 浮点计算指令进入SIMD指令替换浮点计算指令框架进行处理.

d)ALU: 浮点计算指令进入计算逻辑单元中进行处理.

e)Floating-Point Insn: 浮点计算指令.

f)UTSNAME: 计算机系统调用utsname函数判断本地平台(Host)架构.

g)Final Insn: 指令处理完毕,翻译后的指令存储到本地cache.

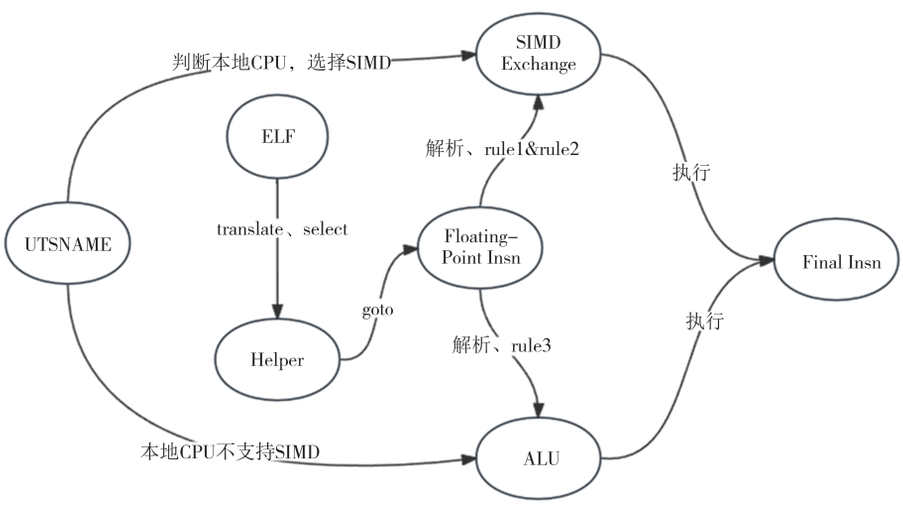

图3　FP-QEMU状态转移图

Fig.3　FP-QEMU state transition diagram

2.1.2　条件定义

a)rule1: 解析读取操作数,判断操作数数值类型是否属于浮点数且精度为双精度(64位)或单精度(32位).

b)rule2: 解析读取操作码,判断运算类型是否属于四则运算.

c)rule3: 操作数不是浮点数、精度超过双精度(64位)范畴、运算不属于四则运算的复杂运算如三角函数等.

2.1.3　动作定义

a)[Floating-Point Insn]→[SIMD Exchange],表明浮点计算指令属于四则运算且精度满足要求时采用SIMD指令替换.

b)[Floating-Point Insn]→[ALU],表明浮点计算指令属于高精度(超过64位精度)/复杂运算时转到ALU浮点计算单元.

c)[UTSNAME]→[SIMD Exchange],表明本地CPU支持SIMD指令且转到具体的SIMD指令实现.

d)[UTSNAME]→[ALU],表明本地CPU不支持SIMD指令且转到ALU浮点计算单元.

2.2　FP-QEMU框架实现

在分析了各种体系架构平台和对应SIMD指令的功能和异同之后,并针对这些异同以及功能深入分析QEMU翻译系统上处理浮点计算时、完整调用SIMD指令来处理的问题和挑战后,本文以最有代表性的X86架构和ARM架构为例,提出当本地平台是X86时,浮点计算部分采用SSE指令来进行处理;本地平台是ARM时,浮点计算部分采用NEON指令来进行处理.本框架适用于一切支持SIMD指令的架构平台.此外,NEON指令中没有关于浮点数除法的指令,本文采用浮点数乘法来替换浮点数除法,也即公式(1)和公式(2)所示:

float32x4_t ax = vrecpeq_f32(axt)　　(1)

float32x4_t bx = vmulq_f32(cx, ax)　　(2)

公式(1)为取axt的倒数赋值给ax,再经过公式(2)运算取cx与ax的乘积实现cx除axt的处理.

以SSE指令替换浮点计算指令为例,首先采用_mm_load_xx指令加载操作数到xmm寄存器,然后判断操作码得到具体操作类型,采用_mm_xx_xx指令进行具体运算操作并将结算结果保存到xmm寄存器中,最后将寄存器结果加载到内存并返回固定格式的值.

2.2.1 SIMD指令融合

将浮点计算和SIMD指令的研究相结合,涉及对浮点计算算法和SIMD并行计算的深入理解,并需要解决两者之间的兼容性和优化问题.本文旨在通过SIMD指令替换浮点计算指令来搭建FP-QEMU框架,通过此框架实现不同本地平台SIMD指令的调用,进而发挥本地平台架构优势,从而提高动态二进制翻译的效率.

如图3所示,翻译ELF文件过程中调用库函数UTSNAME模块获取CPU参数,然后将获得的CPU参数信息(如X86-64和AARCH64等)与预设值比对判断,不同的SIMD指令实现函数已经被提前封装成不同的.h文件保存到FP-QEMU源码中,此时通过switch语句根据不同的判断条件(CPU信息)跳转到不同的.h文件中调用对应的SIMD处理函数(如X86架构CPU调用对应X86_Float.h文件中的SSE指令).图4是FP-QEMU动态二进制翻译系统执行过程中具体函数调用图.

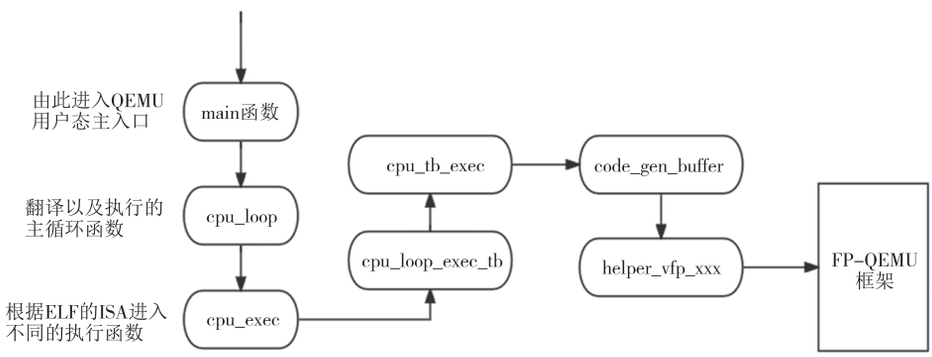

图4 FP-QEMU框架函数调用图

Fig.4 Function call graph of FP-QEMU framework

2.2.2 FP-QEMU实现及其方法描述

结合上文分析QEMU对浮点指令处理的流程,通过调用函数的方式辅助完成指令的功能,函数把所有关于浮点计算的内容由高级语言来模拟,需要冗长的指令来实现目标平台指令,这是十分费时的,代码冗余度也高.此外,原处理方式加入了大量的特殊情况的判断,经过对SPEC2006浮点型课题分析发现,浮点计算的操作数95%以上情况是常规数据,非数、无穷大等情况较少.

针对此情况,本文提出采用SIMD指令替换浮点计算的方法,以达到提高动态二进制翻译效率的同时精简函数的目的.图5所示程序片断是对float64_add的优化示例,其余运算如减、乘、除等运算只需修改对应SIMD指令中的运算指令,其余加载操作数等指令可基本保持不变.此外需注意NEON指令中没有关于浮点数除法的指令,本文采用浮点数乘法来替换浮点数除法,具体过程可见公式(1)和公式(2).

3 实验与结果

3.1 实验环境

本次测试主要在两种体系架构上进行,因此实验测试环境如表1所示.

X86架构的测试平台采用的是8核8线程的Intel i7-9700处理器,是一款CPU主频达到3 GHz的台式机处理器,ARM架构的测试平台采用的是飞腾D2000处理器,集成了8个飞腾自主研发的高性能内核FTC663.本文实验使用的翻译器为QEMU 6.0,通过改良优化成功搭建了FP-QEMU框架,支持SSE指令和NEON指令替换浮点计算指令.

实验采用标准性能测试集SPEC CPU 2006,由于本文工作主要集中在浮点数计算方面,因此SPEC CPU 2006测试用例均为浮点型测试用例.

3.2 实验结果

实验采用未使用SIMD指令替换的翻译方式

```
float64_add(float64 a,float64 b,float_status*s)
{
union_float64 ua, ub, ur;
ua.s=a;
ua.s=b;
if(unlikely(!can_use_fpu(s))) {
goto soft;
}
float64_input_flush2(&ua.s,&ub.s,s);
if(unlikely(!pre(ua,ub))) {
goto soft;
}
ur.h=hard(ua.h,ub.h);
if(unlikely(f64_is_inf(ur))) {
s- > float_exception_flags | =flosat_flag_overflow;
} else if(unlikely(fabs(ur.h) < =DBL_MIN) &&
post(ua,ub)) {
goto soft;
}
return ur.s;
soft;
return soft(ua.sub.s);
}
```

(a)优化前

```
float64_add(float64 a,float64 b,float_status*s)
{
union_float64 ua, ub,;
double ruint;//d定义 float/double
类型返回值变量 ruint
ua.s=a;
ua.s=b;
double af,bf;//d定义 float/double 类型变量 af,bf
float64_input_flush2(&ua.s  &ub.s,s);
af=*(double*)&a;
bf=*(double*)&b;//对af,bf赋值同时进行a,b
的强制类型转换
_m128d s vec=_mm_load)sd(&af);//加载内存中
的变量值到寄存器
_m128d result_vec=
_mm_add_pd(a_vec,_mm_load_sd(&bf));
//选择SIMD指令进行加/减/乘/除计算,
将值从寄存器存到内存中
ruint=(double)resilt vec[0]//强制类型转换
内存中的值并保存到ruint中
return*(floatf64*)&ruint;
}
```

(b)优化后

图5　优化前后的程序片段

Fig.5　Program fragments before and after optimization

表1　二进制翻译本地环境

Tab.1　Binary translation local environment

Items	Platform1	Platform2
OS	Windows 10	UOS V20
CPU	Intel(R) Core(TM) i7-9700CPU@ 3.00 GHz	飞腾 D2000/8@ 2.3 GHz

QEMU 6.0版本和采用SIMD指令替换浮点计算的FP-QEMU框架.QEMU 6.0翻译器翻译时间用Time0来表示,FP-QEMU的翻译时间用Time1表示,完整测试了SPEC 2006中的浮点型课题,由于实验过程中某些课题如435.gromacs等加速效果相对较差,因此在图6和图7的结果展示中未予以实现.实验共测试3轮,取其均值.那么加速比计算公式如公式(3)所示:

$$加速比=(Time0-Time1)/Time0 \tag{3}$$

测试结果如图6和图7所示.

如图6所示,X86架构作为本地平台在使用FP-QEMU框架处理浮点计算之后,所测试浮点型课题的翻译效率均有所提升.其中最高加速比可达51.5%,平均加速比可达37.42%,显著提高了X86架构作为本地平台的翻译效率.

如图7所示,ARM架构作为本地平台在使用FP-QEMU框架处理浮点计算之后,所测试浮点型课题的翻译效率均有所提升.其中最高加速比可达59.59%,平均加速比可达52.72%,显著提高了ARM架构作为本地平台的翻译效率.

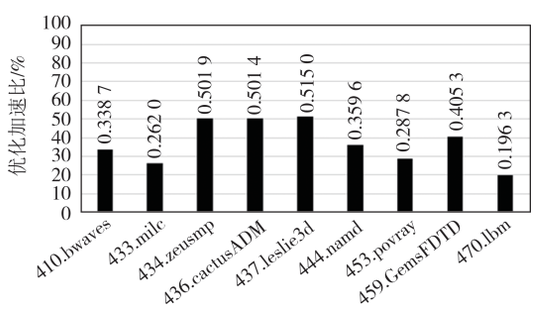

图6　X86架构浮点计算优化加速比

Fig.6　X86 architecture floating-point computation optimization speedup

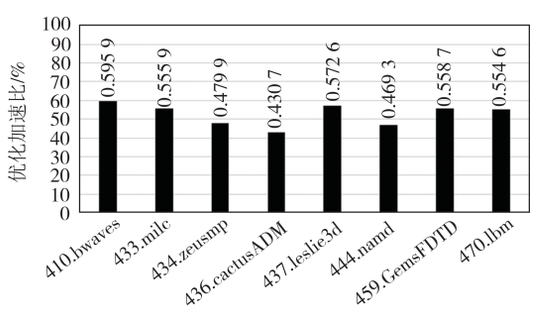

图7　ARM架构浮点计算优化加速比

Fig.7　ARM architecture floating-point computation optimization speedup

根据文献[3]可知,未经优化的二进制翻译器翻译效率通常只有本地编译执行的10%左右.因此本文针对不同浮点型课题做了相对本地执行效率的对比,X86平台和ARM平台相对本地执行的效率对比结果如表2和表3所示.

如表2所示,X86架构机器作为本地实验平台时,未经优化的QEMU版本执行浮点型课题的平均效率只有本地执行的0.70%左右,而经过优化后的FP-QEMU的效率能达到1.12%左右,最高可达2.12%(433)比QEMU平均加速了1.6倍,其中最高加速比达到了2.12倍(437).上述结果有效证实了FP-QEMU系统对X86架构起到了良好的加速效果.

表2 X86架构下QEMU和FP-QEMU相比本地执行效率

Tab.2 Efficiency of QEMU and FP-QEMU on X86 architecture compared with local execution

Items	QEMU/%	FP-QEMU/%
410.bwaves	1.06	1.61
433.milc	1.56	2.12
434.zeusmp	0.44	0.89
436.cactusADM	0.20	0.40
437.leslie3d	0.94	1.93
444.namd	0.59	0.92
453.povray	0.28	0.39
459.GemsFDTD	0.68	1.15
470.lbm	0.53	0.66

如表3所示,ARM架构机器作为本地实验平台时,未经优化的QEMU版本执行浮点型课题的平均效率只有本地执行的0.79%左右,而经过优化后的FP-QEMU的效率能达到1.78%左右,最高可达3.84%(410),比QEMU,平均加速了2.48倍(410),其中最高加速比达到了2.48倍.上述结果有效证实了FP-QEMU系统对于ARM架构同样起到了良好的加速效果.

表3 ARM架构下QEMU和FP-QEMU相比本地执行效率

Tab.3 Efficiency of QEMU and FP-QEMU on ARM architecture compared with local execution

Items	QEMU/%	FP-QEMU/%
410.bwaves	1.55	3.84
433.milc	0.90	2.02
434.zeusmp	0.49	0.94
436.cactusADM	0.26	0.45
437.leslie3d	1.19	2.78
444.namd	0.47	0.88
459.GemsFDTD	0.76	1.72
470.lbm	0.73	1.64

表4是在ARM架构平台上采用FP-QEMU执行浮点型课题的由标量转换为向量的浮点运算型指令占总指令数.

表4 浮点型课题中标量转换为向量的指令占总指令数百分比

Tab.4 Percentage of instructions converted from scalars to vectors in floating-point projects

Items	Ratio/%
410.bwaves	20.94
433.milc	21.57
434.zeusmp	15.91
436.cactusADM	18.19
437.leslie3d	21.51
444.namd	24.45
459.GemsFDTD	15.47
470.lbm	32.03

由以上结果图表分析可知,FP-QEMU系统对于不同课题的加速情况与浮点型课题中标量转换为向量的指令占总指令数百分比成正相关.

3.3 结果分析

综上所述,通过研究SPEC CPU 2006中不同浮点型课题的加速比,可以发现采用FP-QEMU框架对于QEMU翻译的性能有较大的提升.X86架构作为基准平台,平均加速比可达37.42%,是本地执行效率的1.6倍.ARM架构作为基准平台,平均加速比可达52.72%,是本地执行效率的2.25倍,大幅度提高了QEMU的执行效率.以ARM架构为例,根据数据统计,434课题中浮点计算占比约16.01%,433课题中浮点计算占比约21.06%,因此433课题的SIMD指令替换率比434课题要高,从而433课题的加速比要高于434课题.此外,本地计算机平台的性能同样对于最后的统计结果有影响,但本文暂不考虑本地计算机性能不同带来的结果差异.

4 结 论

本文针对动态二进制翻译中浮点计算指令处理翻译模式中的开销,给出了一种优化的处理框架FP-QEMU.FP-QEMU框架充分解析了本地平台SIMD指令的性质和功能并将SIMD指令分析利用,通过不同本地平台采用不同对应SIMD指令替换浮点计算指令的方法达到优化浮点计算进而提高动态二进制翻译效率的目的.通过动态二进制翻译器QEMU的实验验证,本文采用SPEC CPU 2006测试基

准,实验结果表明FP-QEMU框架对动态二进制翻译的优化是有效的.此外,FP-QEMU框架提供了一个清晰的设计思路,研究人员可以根据自己的需求,在该框架的基础上添加新的功能、优化算法或者改进性能.该框架为研究人员提供了一个起点和参考,使他们能够在此基础上开展深入研究,并为未来的相关工作做出更多的贡献.

参考文献

[1] 傅立国,庞建民,王军,等. 动态二进制翻译中库函数处理的优化[J]. 计算机研究与发展,2019,56(8):1783-1791.
FU L G, PANG J M, WANG J, et al. Optimization of library function disposing in dynamic binary translation[J]. Journal of Computer Research and Development, 2019, 56(8): 1783-1791. (in Chinese)

[2] BELLARD F. QEMU, a fast and portable dynamic translator[C]// Proceedings of the Annual Conference on USENIX Annual Technical Conference. Anaheim, CA. ACM, 2005: 41.

[3] 胡伟武,汪文祥,吴瑞阳,等. 龙芯指令系统架构技术[J]. 计算机研究与发展,2023,60(1):2-16.
HU W W, WANG W X, WU R Y, et al. Loongson instruction set architecture technology[J]. Journal of Computer Research and Development, 2023, 60(1): 2-16. (in Chinese)

[4] 石强. 面向国产处理器的二进制翻译关键优化技术研究[D]. 郑州:解放军信息工程大学,2017:19-34.
SHI Q. Research on key optimization technologies of binary translation for the domestic CPU [D]. Zhengzhou: PLA Information Engineering University, 2017: 19-34. (in Chinese)

[5] 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[J]. 国防科技大学学报,2021,43(4):156-162.
LI N, PANG J M. Intermediate code optimization method for binary translation based on intermediate representation rule replacement [J]. Journal of National University of Defense Technology, 2021, 43(4): 156-162. (in Chinese)

[6] HONG D Y, HSU C C, YEW P C, et al. HQEMU: a multi-threaded and retargetable dynamic binary translator on multicores [C]//Proceedings of the Tenth International Symposium on Code Generation and Optimization. San Jose, California. ACM, 2012: 104-113.

[7] WANG Z G, LIU R, CHEN Y F, et al. COREMU: a scalable and portable parallel full-system emulator [C]//Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming. San Antonio TX USA. ACM, 2011: 213-222.

[8] DING J H, CHANG P C, HSU W C, et al. PQEMU: a parallel system emulator based on QEMU [C]//2011 IEEE 17th International Conference on Parallel and Distributed Systems. Tainan, Taiwan, China. IEEE, 2011: 276-283.

[9] POEPLAU S, FRANCILLON A. SymQEMU: compilation-based symbolic execution for binaries [C]//Proceedings 2021 Network and Distributed System Security Symposium. Virtual. Reston, VA: Internet Society, 2021.

[10] ZHAO Z Y, JIANG Z, LIU X M, et al. DQEMU: a scalable emulator with retargetable DBT on distributed platforms [C]// Proceedings of the 49th International Conference on Parallel Processing. Edmonton, AB, Canada. ACM, 2020: 1 - 11.

[11] LUK C K, COHN R, MUTH R, et al. Pin: building customized program analysis tools with dynamic instrumentation [C]// Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. Chicago IL USA. ACM, 2005: 190-200.

[12] ZENG J Y, FU Y C, LIN Z Q. PEMU: a pin highly compatible out-of-VM dynamic binary instrumentation framework [C]// Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul Turkey. ACM, 2015: 147-160.

[13] LI M L, PANG J M, YUE F, et al. Enhancing dynamic binary translation in mobile computing by leveraging polyhedral optimization [J]. Wireless Communications and Mobile Computing, 2021: 6611867.

[14] DOLAN-GAVITT B, HODOSH J, HULIN P, et al. Repeatable reverse engineering with PANDA [C]//Proceedings of the 5th Program Protection and Reverse Engineering Workshop. Los Angeles, CA, USA. ACM, 2015: 1-11.

[15] 李柏举. 在HQEMU系统模拟器的动态二元翻译引擎上产生SIMD指令[D]. 上海:上海交通大学,2012.
LI B J. SIMD instructions generated on the dynamic binary translation engine of the HQEMU system simulator[D]. Shanghai: Shanghai Jiaotong University, 2012. (in Chinese)

[16] LI J H, ZHANG Q, XU S, et al. Optimizing dynamic binary translation for SIMD instructions[C]//International Symposium on Code Generation and Optimization (CGO'06). NewYork, NY, USA. IEEE, 2006.