计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220196
60(1): 2−16, 2023

龙芯指令系统架构技术

胡伟武[1,2,3] 汪文祥[3] 吴瑞阳[3] 王焕东[3] 曾 露[3] 徐成华[3] 高 翔[3] 张福新[1,2,4]
[1](中国科学院计算技术研究所 北京 100190)
[2](中国科学院大学 北京 100049)
[3](龙芯中科技术股份有限公司 北京 100095)
[4](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)
(hww@loongson.cn)

Loongson Instruction Set Architecture Technology

Hu Weiwu[1,2,3], Wang Wenxiang[3], Wu Ruiyang[3], Wang Huandong[3], Zeng Lu[3], Xu Chenghua[3], Gao Xiang[3], and Zhang Fuxin[1,2,4]
[1] (*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2] (*University of Chinese Academy of Sciences*, *Beijing* 100049)
[3] (*Loongson Technology Corporation Limited*, *Beijing* 100095)
[4] (*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

Abstract In this paper, the Loongson instruction set architecture (LoongArch) is introduced, which takes care of both advancement and software compatibility. LoongArch absorbs new features of recent ISA development to improve performance and reduce power consumption. New instructions, runtime environments, system states are added to LoongArch to accelerate binary translation from x86, ARM and MIPS binary code to LoongArch binary code. Binary translation systems are built on top of LoongArch to run MIPS Linux applications, x86 Linux and Windows applications, and ARM Android applications. LoongArch is implemented in the 3A5000 four-core CPU product of Loongson Technology Corporation Limited. Performance evaluation of SPEC CPU2006 with the 3A5000 and its FPGA system shows that, with the same micro-architecture, LoongArch performs on average 7% better than MIPS. With the hardware support, the binary translation from MIPS to LoongArch can be done without performance loss, and that from x86 to LoongArch performs 3.6(int) and 47.0(fp) times better than QEMU system. LoongArch has the potential to remove the barrier between different ISAs and provides a unified platform for a new eco-system.

Key words Loongson CPU; MIPS architecture; LoongArch; binary translation; compatibility; software eco-system

摘 要 介绍了统筹考虑先进性和兼容性要求的龙芯指令系统架构——龙架构(LoongArch). LoongArch 吸纳了近年来指令系统设计领域诸多先进的技术发展成果,易于高性能低功耗的实现和编译优化;融合了各种国际主流指令系统的主要功能特性,不仅能够确保现有龙芯电脑上应用二进制的无损迁移,而且能够实现多种国际主流指令系统的高效二进制翻译.LoongArch 已经被实现于龙芯中科技术股份有限公司研制的 3A5000 四核 CPU.SPEC CPU2006 的实验结果表明,在相同微结构下,LoongArch 性能比龙芯 CPU 原指令系统 MIPS 平均提升超过 7%.在硬件辅助支持下,SPEC CPU2000 程序从 MIPS 翻译到 LoongArch 可以实现无损翻译,其定点程序子集和浮点程序子集从 x86 翻译到 LoongArch 的效率分布达 QEMU 二进制翻译器的 3.6 倍和 47.0 倍.LoongArch 有望消除指令系统之间的壁垒,使得不同指令集的软件能够融合到统一的 LoongArch 平台上,不加区别地高效运行.

收稿日期:2022−03−07; **修回日期**:2022−07−06
基金项目:北京市科委项目(Z211100004421005)
This work was supported by the Program of Beijing Municipal Science & Technology Commission (Z211100004421005).

关键词 龙芯CPU；MIPS架构；龙架构；二进制翻译；兼容；软件生态系统

中图法分类号 TP332

我国的CPU研发应采用兼容指令系统还是自主研发指令系统是学术界和产业界长期争论的一个话题.兼容指令系统的优点是可利用现有的成熟软件生态，缺点是长远发展受制于人且不利于自主软件产业的发展.自主研发指令系统则反之.本文介绍充分考虑兼容需求的龙芯自主研发指令系统架构——龙架构（Loongson instruction set architecture，LoongArch）.龙架构[1]包括基础部分、向量扩展、虚拟化和二进制翻译扩展3个扩展部分，共计近2 000条指令.

LoongArch具有自主设计、技术先进、兼容生态3方面特点.LoongArch从整个架构的顶层规划，到各部分的功能定义，再到细节上每条指令和每个寄存器的编码、名称、含义，全部自主重新设计，具有充分的自主性.龙架构摒弃了传统指令系统中部分不适应当前软硬件设计技术发展趋势的陈旧内容，吸纳了近年来指令系统设计领域诸多先进的技术发展成果，易于硬件的高性能低功耗设计和软件的编译优化以及操作系统、虚拟机的开发.龙架构在设计时充分考虑兼容生态需求，融合了包括x86，ARM在内国际主流指令系统的主要功能特性，同时依托龙芯团队在二进制翻译方面十余年的技术积累创新，不仅能够实现现有龙芯电脑上应用二进制的无损迁移，而且能够实现多种国际主流指令系统的高效二进制翻译.

软件生态是龙芯指令系统架构能否成功的基础和关键.除了迁移BIOS和操作系统内核到LoongArch，还需要3+3+3的主要编译系统.第1个“3”是指3个高级语言编译器，包括GCC，LLVM和GoLang.第2个“3”是指3个重要虚拟机，包括Java，JavaScript和.NET.第3个“3”是指3个二进制翻译系统，包括从MIPS到LoongArch的二进制翻译系统、从x86到LoongArch的二进制翻译系统和从ARM到LoongArch的二进制翻译系统.在上述“3+3+3”主要编译系统的基础上，能够突破指令系统的壁垒，构建LoongArch的软件生态体系.

支持LoongArch的龙芯3A5000处理器[2]芯片已经成功量产.在3A5000上成功运行了完整的Linux操作系统，并通过二进制翻译技术高效运行原有龙芯计算机上的MIPS应用、x86计算机Linux和Windows系统上的各种应用.实测结果表明，在相同微结构的情况下，LoongArch的应用性能普遍优于MIPS，部分应用的性能提高超过40%.SPEC CPU2000从MIPS到LoongArch的二进制翻译效率达90%以上，从x86到LoongArch的二进制翻译定点和浮点效率分别为QEMU[3]的3.6倍和47.0倍.上述测试结果表明，同时实现兼顾自主和兼容的指令系统是可行的，为建立自主可控的信息技术体系和产业生态打下坚实的基础.

1 LoongArch介绍

LoongArch继承了精简指令集计算机（reduced instruction set computer，RISC）的设计传统，其指令长度固定且编码格式规整，大多数指令为三操作数，仅有load/store访存指令可以访问内存.龙架构按照地址空间大小可分为32位和64位2个版本，分别简称为LoongArch32和LoongArch64，LoongArch64应用级向下二进制兼容LoongArch32.下面从指令编码、组成部分和对二进制翻译的支持3个方面对龙架构进行介绍.

1.1 指令编码

LoongArch中的所有指令长度均为32位，且要求指令地址4字节边界对齐，当指令地址不对齐时将触发地址错例外.

LoongArch中指令编码风格规整.所有寄存器操作数域都从第0位开始从低到高依次摆放，操作码都是从第31位开始从高到低依次摆放.如果指令中包含立即数操作数，那么立即数域位于寄存器域和操作码域之间，根据不同指令类型有不同的长度.具体来说，包含9种典型的指令编码格式，即3种不含立即数的编码格式2R，3R，4R，以及6种含立即数的编码格式2RI8，2RI12，2RI14，2RI16，1RI21，I26.图1给出了这9种典型编码格式的具体定义.

1.2 组成部分

LoongArch采用基础部分加扩展部分的模块化组织形式.一个兼容龙架构的CPU，除实现必需的基础部分（Loongson base，LBase）外，可根据实际需求选择实现各扩展部分.目前龙架构已定义的扩展部分包括：虚拟化（Loongson virtualization，LVZ）扩展、二进制翻译（Loongson binary translation，LBT）扩展、128位向量扩展（Loongson SIMD extension，LSX）和256位高级向量扩展（Loongson advanced SIMD extension，LASX）.

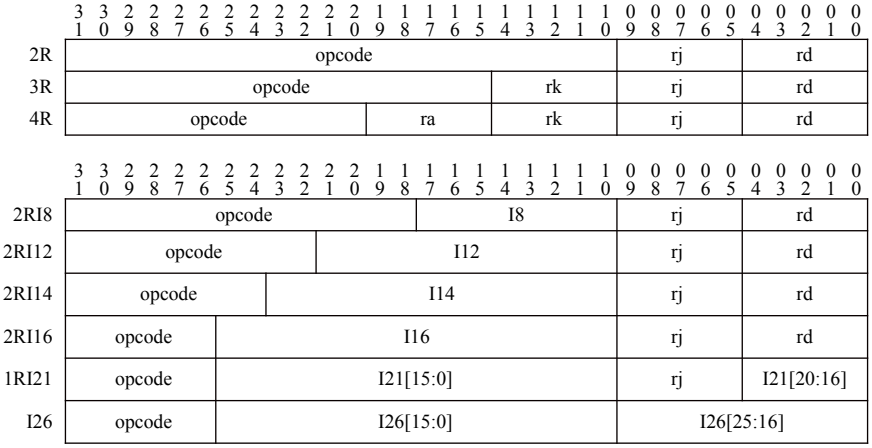


Fig. 1 Instruction encoding format of LoongArch

图 1 LoongArch 指令编码格式

LoongArch 的基础部分包含用户态和核心态 2 方面内容.用户态部分定义了常用的整数和浮点数指令，能够充分支持现有各主流编译系统生成高效的目标代码.核心态部分在处理器特权等级、例外和中断处理、存储管理以及配套的控制状态寄存器等方面给出了明确规范，旨在支持目前主流的类 Unix 操作系统.与龙芯 CPU 原来实现的 MIPS 架构相比，龙芯基础架构充分利用后发优势，摒弃了传统指令系统中部分不适应当前软硬件设计技术发展趋势的陈旧内容，同时积极吸纳了近年来指令系统设计领域诸多先进的技术发展成果，主要特点包括：

1)取消传统 RISC 指令系统中一些过时指令.包括取消陷阱指令和进行溢出判断的运算指令，整数乘除运算的结果将直接写入通用寄存器而非单独的 HI/LO 寄存器.这些过时指令或者不利于指令流水线的高效实现，或者当前主流编程语言已经很少使用.

2)增加一些便于实现且有利于软件提高性能的指令.引入基于 PC 的运算指令并为间接跳转指令添加立即数偏移，同时增加相对 PC 跳转指令的偏移范围，这些调整有利于改善位置无关代码中长跳转和数据访问的指令(序列)的执行效率，并且能够大幅度减小全局偏移表(global offset table, GOT)的规模，从而降低因 GOT 规模过大带来的维护和访问开销.增加原子访存修改指令，解决传统 LL/SC 指令在大规模并发执行情况下失效重试开销急剧增加的问题.

3)取消转移指令延迟槽.对于现代多发射的乱序动态流水线，延迟槽无法起到提升性能的作用，还容易成为实现负担.

4)由硬件负责处理所有流水线的冲突，允许普通访存指令的地址非对齐访问，由硬件维护指令和数据缓存之间的数据一致性.这些功能简化了软硬件的界面，降低了应用迁移的成本.

5)优化处理器特权态、例外系统和计时系统.处理器特权等级从 MIPS 的用户态、监管态和核心态 3 个状态调整为 PLV0, PLV1, PLV2, PLV3 四个状态.例外系统从多个例外公用入口调整为每个例外独立入口.计时系统的计时频率恒定，不再随处理器核频率变化而变动，消除处理器动态功耗管理带来的问题.

6)取消地址空间的固定分段方式以及地址段与特权等级、映射方式间的固定绑定，代之以单一平整(flat)寻址空间且所有存储管理配置信息软件均可动态调整.

7)支持控制寄存器的原子修改、规范核外控制寄存器且使用独立的寻址空间.增加控制寄存器的原子修改支持以简化系统软件实现.大幅度拓展核内控制寄存器寻址空间，为指令系统核心态部分的后续演进解除束缚.规范核外控制器并使用独立的寻址空间，将各种多核多路系统的底层硬件信息封装为固定统一格式呈现给软件.

LoongArch 的虚拟化扩展部分定义了一系列硬件支持特性，旨在提升系统虚拟化实现的性能，涉及处理器虚拟化、内存虚拟化和 I/O 虚拟化 3 个方面.其具体内容主要包括客户机专用的运行模式和特权资源、例外分级处理、2 级地址翻译加速以及中断虚拟化.

LoongArch 的二进制翻译扩展部分引入了一系列硬件特性，进而以软硬件协同的方式大幅度提升跨指令系统二进制翻译执行效率.有关这部分的详细介绍将在本文的 2.2 节展开.

LoongArch 下的向量扩展部分定义了一系列单

指令多数据(SIMD)指令，利用数据级并行性提升程序执行性能.向量扩展部分可进一步分为向量扩展(LSX)和高级向量扩展(LASX)，后者在前者基础上将操作向量位宽从128位增至256位并进一步扩展.

上述基础部分加扩展部分的组成形式对于LoongArch32和LoongArch64架构均适用.从功能角度而言，各部分中属于LoongArch32的内容均包含在LoongArch64中.各部分中的用户态相关内容，LoongArch64向下二进制兼容LoongArch32.

1.3 LoongArch的ABI优化

LoongArch的应用程序二进制接口(application binary interface，ABI)与龙芯CPU原使用MIPS相比对寄存器使用约定进行了优化.表1和表2分别是64位MIPS和LoongArch的通用寄存器使用约定.与MIPS相比，LoongArch有5方面差异：

1)取消了汇编暂存器(at).MIPS的一些汇编宏指令由多条硬件指令合成，汇编暂存器用于数据周转.LoongArch指令系统的宏指令可以不用周转寄存器或者显式指定周转寄存器，汇编暂存器不再必要了，这可以增加编译器可用寄存器的数量.

2)取消了预留给内核的专用寄存器(k0/k1).MIPS预留2个寄存器的目的是支持高效异常处理，可以省去异常处理过程中保存上下文到内存中的开销.LoongArch提供了数据保存状态控制寄存器来高效暂存数据，可以在不预留通用寄存器的情况下保持高效实现，给编译器留下了更多的可用寄存器.

3)取消了gp寄存器.MIPS中用gp寄存器指向GOT以协助动态链接器计算可重定位的代码模块的相关符号位置.LoongArch指令集支持基于PC的运算指令，能够用其他高效的方式实现动态链接，不再需要额外花费一个通用寄存器.

4)复用参数寄存器和返回值寄存器，参数寄存器a0/a1也被用作返回值寄存器.这也是现代指令系统比较常见的做法，它进一步增加了编译器可用的通用寄存器的数量.

5)增加了线程指针寄存器tp，用于高效支持多线程实现，它总是指向当前线程私有存储(thread local storage, TLS)区域.

Table 1 MIPS N64 Integer Register Usage Convention

表1 MIPS N64定点寄存器使用约定

寄存器编号	MIPS N64助记符	使用约定
0	zero	总是为0
1	at	汇编暂存寄存器
2~3	v0,v1	子程序返回值
4~11	a0~a7	子程序的前8个参数
12~15	t0~t3	不需要保存的寄存器
16~23	s0~s7	保存寄存器，子程序使用需要保存和恢复
24~25	t8,t9	寄存器
26~27	k0,k1	为异常处理保留
28	gp	全局指针
29	sp	栈指针
30	s8/fp	保存寄存器，或作为帧指针
31	ra	子程序返回地址

Table 2 LoongArch64 Integer Register Usage Convention

表2 LoongArch64定点寄存器使用约定

寄存器编号	LA64助记符	使用约定
0	zero	总是为0
1	ra	子程序返回地址
2	tp	thread pointer，指向TLS
3	sp	栈指针
4~11	a0~a7	子程序的前8个参数
4~5	v0~v1	v0/v1是a0/a1的别名，用于表示返回值
12~20	t0~t8	不需要保存的寄存器
21	reserved	暂时保留不用
22	fp	frame pointer，栈帧指针
23~31	s0~s8	保存寄存器，子程序使用需要保存和恢复

1.4 LoongArch对二进制翻译的支持

二进制翻译技术是实现跨指令系统兼容的重要手段.二进制翻译技术在宿主机(host)上用软件模拟出一个目标机/客户机(guest)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.如在LoongArch计算机上模拟x86指令系统，从而实现与x86兼容.二进制翻译的最大问题是效率问题，用软件模拟的CPU比硬件直接实现的CPU慢很多，一般有数量级的差异.

通过硬件支持和软硬件协同可以有效提高二进制翻译的效率.LoongArch在认真分析x86, ARM, MIPS, RISC-V指令系统主要特点的基础上，增加了为提高x86, ARM, MIPS, RISC-V二进制翻译性能至关重要的指令功能.下面举例说明LoongArch对x86二进制翻译的具体支持.

1)增加生成运算标志的运算指令以及根据运算标志生成转移条件的指令.x86定义了独立的运算结果标志寄存器EFLAGS，由运算指令在产生结果数值的同时予以更新，转移指令依据EFLAGS的值进行跳转判断.若采用纯软件方式模拟一条x86运算指令

对于 EFLAGS 的更新需要几十条指令.图 2(b) 给出了模拟图 2(a) 中 x86 指令“SUB ECX,EDX”运算结果及常用的四位标志(SF, ZF, OF, CF)的 LoongArch 基础指令序列所需要的近 30 条指令.为了降低软件模拟运算结果标志的二进制翻译开销，LoongArch 定义了一组专门用于产生 x86 运算结果标志的指令，这类指令仅将运算结果标志写入到通用寄存器中.同时，LoongArch 还定义了根据通用寄存器中的运算结果标志值，生成 x86 转移指令相对应的转移条件的指令.图 2(c) 给出了采用二进制翻译扩展的 LoongArch 翻译图 2(a) 中 x86 指令“SUB ECX,EDX”的指令序列，与仅采用基础指令的翻译结果相比指令数大幅度减少.

```
0       SUB     ECX,    EDX
1       JE      X86_target
```

(a) 原始x86代码片段

```
0.00        SUB.W     Result,   Recx,    Redx
0.01        SRLI.W    Rsf,      Result,  31      /*SF为Result[31]*/
0.02        BEQ       Result,   R0,      L1
0.03        ADDI.W    Rzf,      R0,      0       /*ZF为0*/
0.04        B         L2
0.05  L1:   ADDI.W    Rzf,      R0,      1       /*ZF为1*/
0.06  L2:   SRLI.W    Rtmp1,    Result,  31
0.07        SRLI.W    Rtmp2,    Redx,    31
0.08        SRLI.W    Rtmp3,    Recx,    31
0.09        BEQ       Rtmp1,    Rtmp3,   L3
0.10        BEQ       Rtmp2,    Rtmp3,   L3
0.11        ADDI.W    Rof,      R0,      1       /*OF为1*/
0.12        B         L4
0.13  L3:   ADDI.W    Rof,      R0,      0       /*OF为0*/
0.14  L4:   SRLI.W    Rhigh2,   Recx,    16
0.15        SRLI.W    Rhigh1,   Redx,    16
0.16        BEQ       Rhigh2,   Rhigh1,  L5
0.17        BLT       Rhigh2,   Rhigh1,  L7
0.18        B         L6
0.19  L5:   SLLI.W    Rlow2,    Recx,    16
0.20        SRLI.W    Rlow2     Rlow2,   16
0.21        SLLI.W    Rlow1,    Redx,    16
0.22        SRLI.W    Rlow1,    Rlow1,   16
0.23        BLT       Rlow2,    Rlow1,   L7
0.24  L6:   ADDI.W    Rcf,      R0,      0       /*CF为0*/
0.25        B         L8
0.26  L7:   ADDI.W    Rcf,      R0,      1       /*CF为1*/
0.27  L8    ADD.W     Recx,     Result,  R0
1.00        BNE       Rzf,      R0,      LA_target
```

(b) LoongArch基础指令翻译代码片段

```
0.0     SUB.W     Result,   Recx,    Redx    /*生成Sub指令运算结果*/
0.1     X86SUB.W  Reflag,   Recx,    Redx    /*生成Sub指令EFLAGS结果*/
0.2     SETX86J   Rtmp,     Reflag,  EQ      /*根据EFLAGS判断转移条件*/
1.0     BNE       Rtmp,     R0,      LA_target
```

(c) LoongArch含二进制翻译扩展后的翻译代码片段

Fig. 2 Example of x86 EFLAGS translation

图 2 x86 EFLAGS 翻译示例

2)增加 x86 浮点寄存器特殊寻址模式和数据格式的支持.x86 架构的 x87 浮点部件有 8 个浮点寄存器采用栈寻址模式，即 x86 浮点指令码中只有所访问寄存器相对于栈顶的偏移值，其加上浮点状态字中的 TOP 域的值才是要访问的寄存器编号.如果在二进制翻译中动态计算上述浮点寄存器号，每条访问浮点寄存器的浮点指令需额外花费多条指令.为解决这一问题，LoongArch 定义了 3 位的 TOP 寄存器、浮点寄存器的 TOP 寻址模式以及用于操作 TOP 寄存器和切换 TOP 寻址模式的指令.当处于 TOP 寻址模式时，普通浮点指令中浮点操作数的寄存器号都是这些域的值与 TOP 寄存器中的值相加后的结果，从而省去了软件计算实际寄存器号的翻译开销.此外，x87 浮点部件支持 40 位的扩展单精度和 80 位的扩展双精度浮点运算，与 RISC 处理器中 32 位单精度和 64 位双精度不同，为此 LoongArch 定义了将 x87 的浮点数与 LoongArch 的浮点数进行格式转换的指令.

3)扩展 TLB MMU 功能支持客户机虚地址到宿主机实地址的硬件直接翻译.系统级二进制翻译过程中所有客户机访存指令都要 2 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，再将客户机的物理地址作为宿主机的虚地址翻译成宿主机的物理地址.在 QEMU[1] 中，客户机虚地址到客户机物理地址的转换是通过软件模拟完成的.图 3(b)

```
0        MOV     %EAX    4(%ESP)
```

(a) 原始x86程序片段

```
0.1      ADDI.W  Rr11,   Resp,   4           /*计算虚拟地址*/
0.2      SRLI.W  Rr12,   Rr11,   12          /*取虚拟页面基地址*/
0.3      ANDI    Rr13,   Rr12,   0xff        /*计算TLB index, 取页面基地址最低8位*/
0.4      LU12I.W Rtmp,   %hi20(tlb)          /*取TLB首地址*/
0.5      ORI     Rtmp,   Rtmp,   %low12(tlb)
0.6      SLLI.W  Rr14,   Rr13,   3           /*由TLB index计算在表中的偏移量, 每个TLB
                                             条目占8字节*/
0.7      ADD.W   Rr14,   Rr14,   Rtmp        /*TLB条目的地址*/
0.8      LD.W    Rr15,   Rr14,   0           /*取TLB中的虚拟页面基地址*/
0.9      BNE     Rr15,   Rr12,   miss        /*TLB不命中*/

    hit:
0.10     LD.W    Rr15,   Rr14,   4           /*取TLB中的物理页面基地址*/
0.11     ANDI    Rtmp,   Rr11,   0xfff       /*取页内偏移量*/
0.12     ADD.W   Rtmp,   Rtmp,   Rr15        /*形成物理地址*/
0.13     LD.W    Reax,   Rtmp,   0           /*取数据*/

   miss:
0.14     B       tlb_refill                  /*查页表, 反填TLB*/
```

(b) LoongArch基础指令翻译代码片段

```
0.1      SETVPT  guest_va                    /*前缀指令指示后续访存携带客户机虚地址*/
0.2      LD.W    Reax    Resp,   4           /*访存指令通过独立x86 TLB进行虚拟地址转换*/
```

(c) LoongArch含二进制翻译扩展后的翻译代码片段

Fig. 3 Example of x86 memory access instruction translation

图 3 x86 访存指令翻译示例

中给出模拟图 3(a) 中 x86 访存指令时，使用 LoongArch 基础指令模拟将客户机虚地址转化为客户机物理地址所需的指令序列，共有 14 条指令且其中包含 2 条访存.为降低实现开销，LoongArch 的 TLB 可以同时存放“宿主机虚地址=>宿主机物理地址”和“客户机虚地址=>宿主机物理地址”2 种类型的页表项，前一类型的页表项由宿主机操作系统中的普通页表提供，后一类型的页表项由二进制翻译虚拟机中维护的影子页表提供.2 类页表项在填入 TLB 的过程中将被标记上不同的页表类型标识.访存指令在查找 TLB 的过程中，也将根据自身携带的是宿主机虚地址还是客户机虚地址，仅查找页表类型标识与之对应的 TLB 表项.当查找过程发生异常时，不同页表类型标识所触发的 TLB 例外类型及其入口也不同.访存指令所携带虚地址的类型判定，可以采用预设地址空间或访存指令前缀指定 2 种方式.图 3(c) 中给出了采用前缀指令方式判定的翻译指令序列，只需要 2 条指令. 具体的芯片实现还可以选择用未使用的某些虚拟地址位来表示地址空间，这样可进一步省去前缀指令，只需要一条指令.

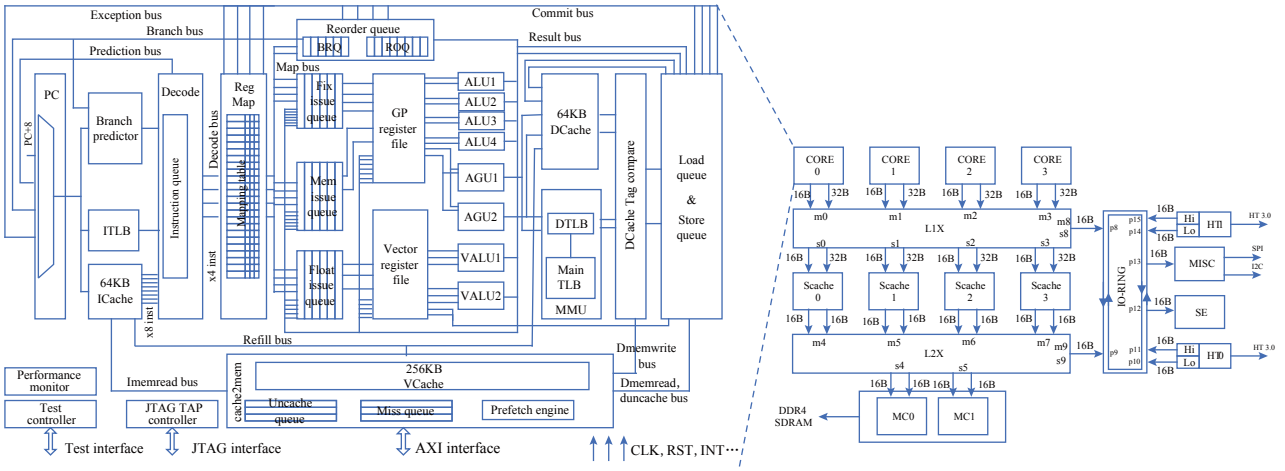
4)增加宿主机寄存器数量，避免用宿主机的内存模拟客户机的寄存器.对客户机寄存器的高效模拟是提高二进制翻译性能的关键.如果宿主机没有足够的寄存器，用内存单元来模拟目标机器的寄存器，就会大幅度降低性能.x86 有 8 个通用寄存器、8 个浮点寄存器和 16 个 256 位的浮点/向量混用寄存器，少于 RISC 架构下通用寄存器、浮点/向量寄存器的个数，可以采用一一对应的模拟方式.在 MIPS 等 RISC 架构中，采用一一对应的模拟方式极容易出现翻译过程中无临时寄存器可用的情况.为此 LoongArch 定义了若干便签寄存器，用于二进制翻译系统临时存储数据.

2 龙芯 3A5000 及其软件生态建设

2.1 龙芯 3A5000 处理器及其计算机系统

龙芯 3A5000 是实现 LoongArch 的第一款龙芯处理器.龙芯 3A5000 通过片上交叉开关集成 4 个 64 位的四发射超标量 LA464 处理器核、16 MB 共享三级缓存、2 个 64 位 DDR4 内存接口、2 个 16 位 HyperTransport3.0 接口.3A5000 使用 12 nm 工艺实现，主频 2.3~2.5 GHz, 内存控制器接口速率为 DDR4-3200, HyperTransport 接口速率为 6.4 Gbps.3A5000 通过 HyperTransport 接口连接龙芯 7A1000 或 7A2000 桥片，并通过桥片连接硬盘、网络、显示器等组成计算机系统.龙芯 3A5000 四核处理器结构如图 4 所示，其中左半边为 LA464 处理器核结构，右半边为 3A5000 处理器顶层结构.

LA464 处理器核采用四发射乱序执行结构. 基本流水线包括 PC、取指、预译码、译码 I、译码 II、寄存器重命名、调度、发射、读寄存器、执行、提交 I 和提交 II 共 12 级.具有 4 个定点运算部件、2 个全功能浮



LA464处理器核结构　　龙芯3A5000处理器顶层结构

Fig. 4　Architecture of 3A5000 and LA464 processor core

图 4　3A5000 及 LA464 处理器核架构

点运算部件、2 个访存部件.重排序缓存为 128 项，定点、浮点、访存 3 个发射队列各为 32 项.定点和浮点物理寄存器堆各 128 项.一级指令缓存和一级数据缓存均为 4 路组相联，各 64 KB，二级牺牲(Victim)缓存为 16 路组相联，共 256 KB，缓存行大小均为 64 B.支持非阻塞存储访问的访存部件的 Load 队列 64 项，Store 队列 48 项，每个处理器核具有 16 项失效队列，用于处理核内缓存失效访问的访存请求.LA464 使用基于目录(directory)的 MESI 缓存一致性(cache coherency)协议.

2.2　龙芯 3A5000 的软件生态建设

软件生态是 LoongArch 能否成功的基础和关键.龙芯 3A5000 的软件生态建设以 Loongnix Linux 社区发行版[4]构建为抓手，以前述“3+3+3”的主要编译系统为核心，经过约 3 年的时间构建了 LoongArch 架构的基础软件生态.

Loongnix 是支持 LoongArch 架构的开源社区版 Linux 操作系统.基于开源的 debian 10 发行版[5]移植适配的 Loongnix 在构建过程中共计完成 24 300 多个操作系统源码包、58 100 多个操作系统二进制包，与架构相关的适配源码量超过 40 万行.

“3+3+3”的主要编译系统是生态建设的核心.GCC，LLVM 和 GoLang 三个编译器，以及 Java，JavaScript 和.NET 三个虚拟机能够基本满足典型 Linux 发行版中软件包迁移以及新应用开发的需求.而 3 个二进制翻译系统则为复用已有主流架构的软件提供了有效渠道.

3A5000 基础生态构建大致可分为 4 个阶段：第 1 个阶段是 GCC 交叉编译工具链与模拟器研发，达到可生成 LoongArch 架构代码并在模拟器上运行相关代码的目标；第 2 个阶段是构建最小 Linux 系统，完成 BIOS、内核开发，迁移常见的 Linux 命令行工具，生成初始根文件系统；第 3 个阶段是更多软件的移植，包括 GCC 之外的各编译系统以及 Chrome 浏览器、KVM 和 Docker 等关键基础软件；第 4 个阶段是上游社区建设与第三方软件适配.前 3 个阶段分别耗时约 6 个月，初步形成了一个较为完整的基于开源软件的生态体系.第 4 个阶段在大约 18 个月的时间内，使得 LoongArch 生态基本达到了龙芯原有 MIPS 生态的成熟程度：进行了大量软硬件磨合优化工作，通过二进制翻译系统实现了一批常见 Windows 应用的高效兼容运行，完成了常用的国产操作系统、办公软件、云和安全软件等重要商业软件的适配，实现了基于 LoongArch 的产品批量销售.

这 4 个阶段总体上应用了 3 个策略：先模拟后真机、先开源后商业、开源开放.在实现新指令系统的 CPU 产品化之前，只能通过模拟器进行基础软件栈的研发和技术验证.前 2 个阶段和第 3 个阶段的一部分都是在模拟环境中进行的，我们设计了能够精确模拟真实硬件的模拟环境，并采用二进制翻译优化技术提升其运行效率以满足大规模软件开发的需求.而蓬勃发展的开源软件运动，则极大地降低了构建基础生态的门槛，使得 LoongArch 能够在较短时间内初步形成软件基础生态.龙芯团队采用开源开放的模式维护 LoongArch 的软件基础生态，有效促进了第三方商业软件的迁移适配.

龙芯 3A5000 的软件生态构建实践表明，在深入掌握“3+3+3”的主要编译系统技术的基础上，充分利

用开源软件生态，在较短的时间内构建一个新指令系统的生态是可行的.

3 龙芯体系结构翻译器

通过跨指令系统的二进制翻译来丰富宿主机的生态是业界常用的手段. 自 20 世纪 90 年代开始，伴随着体系结构的多样化浪潮，多种宿主机指令系统均针对 x86 目标机提出了各自的二进制翻译平台，例如，Digital FX!32[5] 用于 Alpha 宿主机系统，IA-32 EL[6] 用于安腾（Itanium）宿主机，Transmeta CMS[7] 用于 Transmeta VLIW 指令系统.此外，IBM 公司以 Power 体系结构为目标机，提出了 DAISY[8] 二进制翻译系统，以支持其 IBM VLIW 指令系统的推广.上述二进制翻译系统能够实现较高的执行效率，例如，FX!32 在 500 MHz Alpha 处理器上达到了 200 MHz x86 处理的性能，IA32 EL 在执行部分 SPEC CPU2000 定点测试时能够平均达到本地编译执行效率的 60%.

移动计算和云计算使得计算平台由传统的单一指令系统向多指令系统发展，催生了计算机系统对多种目标机的适应性的新需求.QEMU 是一种广泛应用的可移植二进制翻译系统，能够适用于多种目标机（如 x86, PowerPC, ARM, Sparc and MIPS）和宿主机. 然而，QEMU 的可移植性以大量牺牲模拟效率为代价，未经优化的 QEMU 模拟执行速度经常不到本地编译执行速度的 10%.其根本原因是 QEMU 为实现对多目标机和多宿主机的兼容，使用体系结构无关的中间代码，从而无法利用宿主机体系结构特征，难以对翻译代码进行深入优化.为此，文献 [9-11] 的研究工作针对特定目标机，利用宿主机特性来优化二进制翻译性能.在推出 LoongArch 之前，龙芯平台曾通过 MIPS 指令系统的 UDI（user defined interface）接口扩展指令，弥补 x86 和 MIPS 指令的语义差距，加速 x86 目标机程序的翻译执行效率，基于 SPEC CPU2000 测试实现了 5~10 倍的性能提升[12-14]，其中部分技术被应用于 LoongArch 中.

作为一种新型指令系统，在其原生软件生态成熟之前，LoongArch 将借助二进制翻译实现与 x86, ARM, MIPS 等指令系统的软件二进制兼容来弥补其早期软件生态的不足，构建从 MIPS 到 LoongArch 的体系结构翻译器 LATM（Loongson architecture translator from MIPS）、从 x86 到 LoongArch 的体系结构翻译器 LATX（Loongson architecture translator from x86）、从 ARM 到 LoongArch 的体系结构翻译器 LATA（Loongson architecture translator from ARM）.这些翻译器支持同一操作系统的用户程序翻译（如 Linux/LoongArch 上运行 Linux/x86 的应用）、不同操作系统的应用翻译（Linux/LoongArch 上运行 Windows 或者安卓应用），以及操作系统本身（直接运行 Windows 或者安卓系统），以达到消除单一指令系统壁垒的效果.这不仅要考虑普通用户程序指令的翻译，还要考虑各种体系结构资源的翻译（如特权资源模拟、虚拟地址转化等），因此我们称之为体系结构翻译器.LoongArch 体系结构翻译器和前人工作的区别在于 LoongArch 定义时就充分考虑多种指令系统二进制翻译的需求，强调更加紧密的软硬件协同，大幅提高了跨指令系统二进制翻译的效率.图 5 给出了支持应用级二进制翻译的 LoongArch 软件生态结构.

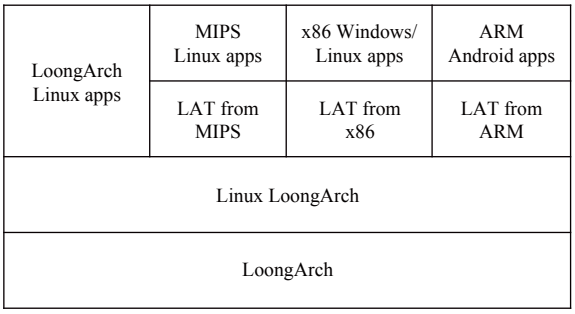


Fig. 5 LoongArch software eco-system

图 5 LoongArch 软件生态结构

3.1 从 MIPS 到 LoongArch 的翻译

由于 MIPS 与 LoongArch 都是 RISC 指令集，绝大多数常见指令都可以实现一对一翻译或者二对二翻译（例如装载 32 位立即数，由于 2 者立即数域大小不同无法保证一对一翻译，但可以把 2 条 MIPS 指令翻译为等效的 2 条 LoongArch 指令），仅有少数指令由于架构差异（如 LoongArch 取消了 MIPS 中转移指令延迟槽）需要多条指令翻译.LATM 实现了 MIPS 到 LoongArch 的寄存器一一映射，并利用 LBT 扩展提供的便签寄存器处理延迟槽中寄存器依赖以及管理运行时环境，避免不必要的寄存器保存恢复.

对于直接跳转的分支指令，通过运行时的翻译块链接可实现高效翻译，采用热点路径探测、基本块代码重排等技术进一步消除额外的跳转翻译开销.而间接跳转的翻译相对复杂，LATM 对函数调用、函数返回以及 switch 语句中的跳转指令进行分别优化.采用跳转目标地址内联技术、多级跳转目标缓存等技术优化函数调用；采用影子栈、返回块预翻译等基础实现函数返回的跳转指令翻译；采用跳转表预翻译

技术优化 switch 语句中的跳转指令.

在运行时环境方面，LATM 实现了高效的系统调用支持，减少翻译系统调用的上下文切换开销，以及多架构库调用等优化，进一步提高翻译效率.

3.2 从 x86 到 LoongArch 的翻译

LATX 在 LATM 的基础上，需要额外处理 EFLAG、浮点栈等问题.LATX 利用了 LBT 扩展的 x86 运算模拟指令、分支模拟指令以及浮点栈模式，大幅降低了对应指令的翻译开销.此外，LATX 通过指令流分析消除不必要的 EFLAG 计算，并针对指令序列中的特定组合按照语义进行二对一的翻译，进一步提高翻译效率.

3.3 不同操作系统应用的翻译

大多数商业二进制翻译系统只支持相同操作系统不同指令集的应用程序，这限制了它们的适用范围.例如，目前产业界对在国产 Linux 操作系统上支持 Windows 应用和驱动程序有强烈的需求，但这类技术无法满足这些需求.支持不同操作系统的应用翻译需要同时支持 2 个层次的翻译：指令集翻译和操作系统 API 翻译.开源软件 Wine[15] 是一个操作系统 API 翻译软件，它可以在 Linux 上用 Linux 的系统调用来模拟实现 Windows 的系统调用，从而实现在 Linux/x86 上运行 Windows/x86 的应用程序.我们通过结合二进制翻译器和 Wine 来实现在 Linux/LoongArch 上运行 Windows/x86 的应用程序.在此基础上，我们还通过 Wine 的本地化提高 API 翻译效率，通过扩充 Wine 实现了部分 Windows 驱动程序的支持，使得大批缺乏 Linux 驱动的打印机能够被用于国产电脑.

3.4 体系结构资源的翻译

要实现不同指令系统的操作系统等系统软件的兼容，就需要实现体系结构资源的翻译或者虚拟.随着云计算技术的发展，现代指令系统大都已经支持将一台计算设备虚拟成多台虚拟设备，而且能够实现很高的虚拟效率.但是高效的异构虚拟化仍然是一个很大的挑战.例如,一个近年来比较活跃的异构虚拟化研究系统 Captive[16]，其效率大约只有原生执行的 1/7.目前在 LoongArch 上，我们已经实现了 LATX-Sys 的原型系统，并在关键的内存虚拟化等环节进行了一些研究[17]，有效地提升了翻译效率.同时，我们正在探索直接在裸机上构建虚拟机管理器，通过消除不必要的宿主机系统软件开销来进一步提升体系结构翻译的效率.

4 性能测试与分析

4.1 基础性能测试

我们用 3 个常见的基准程序来测试 LoongArch 架构首款处理器龙芯 3A5000 的基础性能表现，包括广泛用于衡量 CPU 计算性能的 SPEC CPU2006[18]、用来测试内存带宽的 Stream 程序[19]、测试系统综合性能的 UnixBench[20].

为了方便比较，同时对比测试了采用 MIPS 架构的 3A4000. 3A4000 和 3A5000 的处理器核均采用相同的四发射 64 位微结构. 3A4000 的主频为 1.8 GHz，三级缓存容量为 8 MB，内存速率为 DDR4-2133.3A5000 的主频为 2.5 GHz，三级缓存容量为 16 MB，内存速率为 DDR4-3200.2 款机器的软件版本保持基本一致，其中编译器版本均为 GCC 8.3.

表 3 给出了 3A4000 和 3A5000 的 SPEC CPU2006 Speed 模式(单核)和 Throughput 模式(四核)下的分值情况，2 款机器的编译器均为 GCC 8.3，采用对所有程序一致的基础优化选项，无条件使能向量优化(对部分程序产生负优化)，未使能自动并行化.3A4000 的编译选项为：-Ofast -mabi=64 -march=loongson3a -mtune=loongson3a -mloongson -sx -mloongson -asx -static -flto，3A5000 的选项为-Ofast -mabi=lp64 -march=loongarch64 -mtune=loongarch64 -mlsx -mlasx -ftree -vectorize -static -flto. 可以看到，3A5000 的定点和浮点性能均大幅度高于 3A4000，而且平均提升幅度大于频率的提升幅度(后者约 39%).

表 4 给出了 3A4000 和 3A5000 的 Stream 带宽测试结果.从表 4 可以看到，3A5000 测试机的 Stream 带宽显著高于 3A4000，这主要得益于更高的内存频率和内存控制器的一些优化措施.

表 5 是 3A4000 和 3A5000 的 UnixBench 分值比较，3A5000 单线程和四线程提升均超过 100%. 可以看出，在微结构相似的情况下，3A5000 比 3A4000 的 UnixBench 性能提升不仅远超过主频提升的幅度，也超过了 SPEC CPU 性能的提升幅度，一个重要原因是 LoongArch 提供了更多的软硬件协同优化空间，使得 Linux 操作系统能够得到更好的优化.

4.2 LoongArch 和 MIPS 的直接性能比较

4.1 节中 3A4000 和 3A5000 的性能比较结果受到指令系统以及其他差异因素(如主频、内存频率等)的影响，无法直接反映指令系统的性能差异.为此，我们通过 3A5000 的 FPGA 平台来实现指令系统的直接

Table 3 SPEC CPU2006 Performance Comparison of 3A4000 and 3A5000

表 3 3A4000 和 3A5000 的 SPEC CPU2006 性能对比

SPEC2006 程序	3A4000 单核分值	3A5000 单核分值	提升幅度/%	3A4000 四核分值	3A5000 四核分值	提升幅度/%
400.perlbench	14.4	25.9	79.7	52.7	94.4	79.2
401.bzip2	10.7	17.2	59.9	35.9	66.1	84.0
403.gcc	12.4	21.3	71.3	40.6	69.9	72.2
429.mcf	12.7	27.2	115.0	30.0	49.6	65.4
445.gobmk	15.3	23.9	56.2	58.7	91.8	56.3
456.hmmer	24.0	42.9	78.9	87.0	169.1	94.4
458.sjeng	14.0	21.3	52.3	55.1	84.1	52.5
462.libquantum	31.8	58.6	84.7	55.6	89.2	60.5
464.h264ref	21.2	34.7	63.5	82.6	137.7	66.6
471.omnetpp	9.1	16.7	83.4	24.5	43.5	77.9
473.astar	10.7	18.0	68.4	33.4	54.6	63.6
483.xalancbmk	14.6	28.2	93.7	37.0	69.8	88.5
定点几何平均	14.9	26.0	74.8	46.0	78.8	71.3
410.bwaves	39.9	57.8	45.0	52.8	84.5	60.0
416.gamess	14.5	23.1	58.6	58.0	91.7	58.2
433.milc	11.5	17.2	48.8	31.1	44.8	43.9
434.zeusmp	16.2	28.8	77.8	56.3	94.0	67.1
435.gromacs	10.4	15.4	47.7	40.8	61.0	49.4
436.cactusADM	31.8	58.4	83.5	78.9	130.0	64.8
437.leslie3d	23.8	39.1	64.2	41.9	72.5	73.0
444.namd	13.7	19.7	44.1	54.1	78.2	44.4
447.dealII	23.8	36.9	55.4	85.4	133.2	55.9
450.soplex	14.1	26.0	84.4	36.7	58.8	60.4
453.povray	21.8	34.2	56.8	86.8	136.0	56.6
454.calculix	11.3	23.7	109.6	40.8	90.5	121.4
459.GemsFDTD	18.3	30.0	63.5	33.0	54.7	65.6
465.tonto	16.1	27.9	73.3	59.9	106.9	78.5
470.lbm	18.5	31.9	72.5	35.9	51.5	43.3
481.wrf	15.0	33.1	121.1	50.6	99.7	97.0
482.sphinx3	18.9	33.9	79.2	41.1	83.0	101.9
浮点几何平均	17.6	29.7	68.7	49.5	82.1	65.9

Table 4 Stream Bandwidth Comparison of 3A4000 and 3A5000

表 4 3A4000 和 3A5000 的 Stream 带宽对比

Stream	3A4000 单核	3A5000 单核	提升幅度/%	3A4000 四核	3A5000 四核	提升幅度/%
Copy	9 466.9	27 649.1	192.1	17 056.7	39 896.4	133.9
Scale	10 438.0	15 219.1	45.8	18 059.9	25 073.3	38.8
Add	10 033.4	18 032.4	79.7	15 658.6	29 768.5	90.1
Triad	10 060.9	17 883.1	77.7	15 977.2	29 146.8	82.4

对比分析.3A5000 的 FPGA 平台在保持微结构及主要接口速率不变的情况下分别实现了 LoongArch 和 MIPS2 套指令系统，主频为 20 MHz. 在 FPGA 平台上运行 SPEC CPU2006 的 train 数据集，使用 GCC-8.3 版编译器，编译选项同 4.1 节.表 6 列出了指令系统为 LoongArch 和 MIPS 时所花费的处理器流水线周期数和指令数，以及 MIPS 相对 LoongArch 的周期数和指令数的比例.

Table 5 UnixBench Comparison of 3A4000 and 3A5000

表 5 3A4000 和 3A5000 的 UnixBench 对比

基准程序	3A4000 单线程	3A5000 单线程	提升幅度/%	3A4000 四线程	3A5000 四线程	提升幅度/%
Dhrystone 2 using register variables	1 373.3	2 707.2	97.1	5 473.9	10 664.2	94.8
Double-Precision Whetstone	425.8	801.9	88.3	1 703.5	3 156.1	85.3
Execl Throughput	545.1	1 183.6	117.1	2 014.3	4 334.7	115.2
File Copy 1 024 bufsize 2000 maxblocks	1 094.2	2 321.6	112.2	1 288.9	3 212.9	149.3
File Copy 256 bufsize 500 maxblocks	688.9	1 541.3	123.7	795	2 022.6	154.4
File Copy 4 096 bufsize 8 000 maxblocks	1 988.1	4 197.4	111.1	2 926.4	6 794.8	132.2
Pipe Throughput	644.3	1 324.7	105.6	2 566	5 160.1	101.1
Pipe-based Context Switching	294.5	808.9	174.7	1 435.6	3 025.1	110.7
Process Creation	469.4	948.6	102.1	1 181.8	3 300.8	179.3
Shell Scripts (1 concurrent)	1 370.5	2 650.7	93.4	3 410.3	6 175.6	81.1
Shell Scripts (8 concurrent)	2 955.3	5 300.6	79.4	3 088.9	6 613.4	114.1
System Call Overhead	508.3	1 435.3	182.4	1 812.5	4 247.1	134.3
基准程序几何平均得分	816.4	1 743.9	113.6	2 022.4	4 432.9	119.2

从表 6 可以看出，MIPS 比 LoongArch 的 SPEC CPU2006 定点程序平均动态指令数和周期数分别多 12.35% 和 8.02%，浮点则分别多 5.57% 和 7.53%.差距最大的定点程序 483.xalancbmk MIPS 比 LoongArch 的指令数和周期数分别多 45.20% 和 45.16%，浮点程序 481.wrf 则分别是 30.45% 和 23.40%. 定点程序中，仅有一个程序 456.hmmer 的动态指令数 LoongArch 略多于 MIPS，而且它的性能依然高于 MIPS. 浮点程序有 5 个 LoongArch 动态指令数多于 MIPS，其中 2 个性能略低于 MIPS. 经分析，这些程序表现不如 MIPS 的一个主要原因是 LoongArch 编译器还没有充分利用乘加指令，在一些能够用乘加指令的地方生成了乘法和加法的组合指令.

在完全相同的微结构情况下，LoongArch 能够有这样的优势要归功于其更合理的指令设计和更高效的 ABI 约定.目前，LoongArch 的编译器尚未得到充分的优化，后续 LoongArch 相对 MIPS 的性能优势还可能进一步增加.

4.3 Loongnix 操作系统性能优化

Loongnix 操作系统针对 LoongArch 架构设计的特点进行了一些优化.例如，避免上下文切换时不必要的保存和恢复操作以降低上下文切换的开销；利用 LoongArch 向量指令优化 memcpy 等函数；利用 LoongArch 提供的更丰富的原子指令支持优化锁和各种同步原语，有助于降低上下文切换和系统调用等开销；利用 LoongArch 进入内核态自动关闭中断的特点省略软件关中断的处理；等等.

表 7 给出了结合 LoongArch 特点优化前后的 UnixBench 分值，优化前 3A5000 平台的 UnixBench 初始分值为单线程 1 438.8，四线程 3 698.4，经过一轮优化后分别提升到 1 743.9 和 4 432.9，提升幅度均为 20% 左右.测试数据表明，LoongArch 指令系统的设计能够有效提升操作系统性能.

4.4 LoongArch 体系结构翻译器性能

4.4.1 LATM 应用翻译性能

表 8 显示了 20MHz FPGA 上 SPEC CINT2000 的 LATM 应用翻译性能，其中第 2 列为采用 MIPS 指令系统原生运行的运行时间，第 3 列为采用 LoongArch 指令系统翻译运行 MIPS 二进制的时间，第 4 列为采用 LoongArch 指令系统原生运行的时间.Perlbmk 和 eon 运行时包括几个不同的输入子集，程序的每一次运行都被当做一个单独的测试.表 8 中数据为运行时间，与速度成反比，可以看出，在 LoongArch 上翻译运行 MIPS 二进制的平均速度达到了原生 LoongArch 的 91%，超过了原生 MIPS.

LoongArch 能够比较高效地翻译运行 MIPS 二进制，主要有 2 方面的原因:1)两者均为 RISC 指令系统，比较相似.SPEC CPU2000 中，90% 以上的 MIPS 指令都能一对一翻译为一条 LoongArch 指令，还有部分指令可以成对翻译为一对 LoongArch 指令.2)LoongArch 为二进制翻译提供的便签寄存器等硬件支持能够有效降低翻译开销.

4.4.2 LATX 应用翻译性能

LATX 的初步性能数据如表 9 所示，在 3A5000

Table 6 Performance Comparison of LoongArch and MIPS SPEC CPU2006

表 6 LoongArch 和 MIPS 的 SPEC CPU2006 性能比较

程序	LoongArch		MIPS		$\frac{\text{MIPS}}{\text{LoongArch}}$/%	
	周期数	指令数	周期数	指令数	指令数	周期数
400.perlbench	5.44E+10	1.35E+11	5.80E+10	1.46E+11	106.66	107.44
401.bzip2	1.11E+11	1.95E+11	1.13E+11	2.00E+11	102.55	102.45
403.gcc	2.10E+09	3.40E+09	2.18E+09	3.93E+09	104.10	115.61
429.mcf	4.59E+10	1.67E+10	4.67E+10	2.14E+10	101.64	128.20
445.gobmk	2.07E+11	3.30E+11	2.15E+11	3.48E+11	103.94	105.43
456.hmmer	5.88E+10	1.82E+11	6.83E+10	1.75E+11	116.22	96.19
458.sjeng	2.97E+11	5.75E+11	3.04E+11	5.85E+11	102.19	101.66
462.libquantum	3.46E+09	7.24E+09	3.45E+09	8.46E+09	99.65	116.81
464.h264ref	1.71E+11	4.82E+11	1.75E+11	5.10E+11	102.25	105.94
471.omnetpp	9.45E+10	1.88E+11	1.04E+11	2.16E+11	110.37	114.73
473.astar	2.09E+11	1.89E+11	2.12E+11	2.05E+11	101.57	108.55
483.xalancbmk	1.06E+11	2.11E+11	1.53E+11	3.06E+11	145.16	145.20
定点算术平均	1.13E+11	2.10E+11	1.21E+11	2.27E+11	108.02	112.35
410.bwaves	4.01E+10	6.29E+10	4.29E+10	6.44E+10	106.82	102.42
416.gamess	2.77E+11	9.00E+11	2.83E+11	8.86E+11	102.23	98.46
433.milc	4.20E+10	1.95E+10	3.96E+10	1.75E+10	94.47	89.86
434.zeusmp	4.93E+10	6.29E+10	5.66E+10	7.32E+10	114.76	116.53
435.gromacs	2.49E+11	3.23E+11	2.49E+11	3.10E+11	100.23	95.94
436.cactusADM	2.95E+10	2.88E+10	3.13E+10	2.73E+10	106.19	95.10
437.leslie3d	7.88E+10	9.00E+10	8.40E+10	9.44E+10	106.53	104.81
444.namd	2.73E+10	4.59E+10	2.75E+10	4.69E+10	100.52	102.29
447.dealII	4.62E+10	8.07E+10	4.80E+10	8.72E+10	103.95	108.12
450.soplex	1.17E+10	1.52E+10	1.20E+10	1.66E+10	102.65	109.10
453.povray	1.28E+10	2.65E+10	1.32E+10	2.90E+10	103.03	109.63
454.calculix	2.72E+09	4.70E+09	3.24E+09	5.18E+09	119.09	110.36
459.GemsFDTD	6.64E+10	6.07E+10	6.58E+10	6.46E+10	99.16	106.52
465.tonto	3.46E+11	8.62E+11	3.75E+11	9.12E+11	108.26	105.79
470.lbm	9.43E+10	7.50E+10	1.03E+11	7.40E+10	109.68	98.64
481.wrf	9.12E+10	1.60E+11	1.13E+11	2.08E+11	123.40	130.45
482.sphinx3	1.77E+10	2.67E+10	2.25E+10	2.95E+10	126.98	110.60
浮点算术平均	8.72E+10	1.67E+11	9.23E+10	1.73E+11	105.57	107.53

上翻译运行 SPEC CPU2000 定点和浮点效率分别是 65.5% 和 70.3%,分别是 QEMU-4.2.1 的 3.6 倍和 40.7 倍. 表 9 中输入数据集为 ref, 编译优化采用对所有程序一致的基础优化(-O3 -static).

5 结论及未来工作

本文介绍了龙芯指令系统架构 LoongArch 的设计及其生态建设实践经验.测试结果表明, 由于吸纳了近年来指令系统设计领域诸多先进的技术发展成果, LoongArch 比龙芯 CPU 原实现的 MIPS 指令系统性能更好; 通过软硬结合的二进制翻译, 可以大幅度提高二进制翻译性能, 高效兼容已有软件生态.实践证明, 在掌握三大 C 编译器(GCC, LLVM, GoLang)、三大虚拟机(Java, JavaScript, .NET)和三大指令系统(MIPS, x86, ARM)的二进制翻译系统的基础上, 可以在较短时间内构建良好的软件生态.

CPU 性能的改进和软件生态的优化都需要持续改进.本文后续的工作包括: 对 LoongArch 指令系统进行更加细致的实验分析, 验证相关指令扩展及设

Table 7 Optimization of 3A5000 UnixBench

表 7 3A5000 的 UnixBench 优化

基准程序	单线程优化前	单线程优化后	提升幅度/%	四线程优化前	四线程优化后	提升幅度/%
Dhrystone 2 using register variables	2 702.2	2 707.2	0.2	10 636.6	10 664.2	0.3
Double-Precision Whetstone	801.8	801.9	0.0	3 150.7	3 156.1	0.2
Execl Throughput	1 101.9	1 183.6	7.4	3 818.3	4 334.7	13.5
File Copy 1 024 bufsize 2000 maxblocks	1 867.9	2 321.6	24.3	2 585.3	3 212.9	24.3
File Copy 256 bufsize 500 maxblocks	1 215.0	1 541.3	26.9	1 638.4	2 022.6	23.4
File Copy 4 096 bufsize 8 000 maxblocks	3 606.4	4 197.4	16.4	5 796.2	6 794.8	17.2
Pipe Throughput	930.5	1 324.7	42.4	3 631.2	5 160.1	42.1
Pipe-based Context Switching	461.5	808.9	75.3	2 183.6	3 025.1	38.5
Process Creation	748.4	948.6	26.8	2 119.4	3 300.8	55.7
Shell Scripts (1 concurrent)	2 470.9	2 650.7	7.3	5 851.0	6 175.6	5.5
Shell Scripts (8 concurrent)	4 936.7	5 300.6	7.4	6 172.0	6 613.4	7.2
System Call Overhead	1 027.2	1 435.3	39.7	3 435.2	4 247.1	23.6
基准程序几何平均得分	1 438.8	1 743.9	21.2	3 698.4	4 432.9	19.9

Table 8 Runtime Comparison of SPEC CINT2000 Train

表 8 SPEC CINT2000 Train 的运行时间比较

测试程序	MIPS/s	LATM/s	LoongArch/s	$\frac{\text{LoongArch}}{\text{LATM}}$/%	$\frac{\text{MIPS}}{\text{LATM}}$/%
164.gzip	2 063.2	1 991.8	1 695.1	85	104
300.twolf	691.2	670.9	549.0	82	103
255.vortex	416.1	431.9	437.5	101	96
176.gcc	156.4	182.1	161.5	89	86
253.perlbmk	1 013.6	1 007.5	922.6	92	101
253.perlbmk	1 032.3	813.6	634.3	78	127
253.perlbmk	613.0	697.0	618.6	89	88
186.crafty	765.9	789.6	760.5	96	97
256.bzip2	1 292.2	1 196.7	1 413.8	118	108
252.eon	38.2	39.5	32.1	81	97
252.eon	53.8	54.5	46.1	85	99
252.eon	206.7	214.3	179.1	84	96
197.parser	393.8	394.4	354.2	90	100
181.mcf	1 474.5	1 510.2	1 553.7	103	98
175.vpr	575.0	570.3	466.4	82	101
175.vpr	605.5	529.5	502.9	95	114
254.gap	295.0	309.1	300.8	97	95
算术平均	687.4	670.7	625.2	91	101

计优化的量化效果; 针对 LoongArch 指令系统的特点进一步优化基础软件来提升性能; 根据进一步测试和应用的结果发现新的性能瓶颈, 持续完善指令系统; 实现基于 Android 的从 ARM 到 LoongArch 的二进制翻译系统; 争取再经过 1~2 轮的软硬件磨合迭代, 到 2025 年前后通过技术手段消除单一指令系统的壁垒, 使得不同指令集的系统及应用软件能够融合到统一的 LoongArch 平台上, 不加区别地运行; 对外开放 LoongArch, 鼓励更多学术界和产业界人士参与架构的持续改进.

作者贡献声明: 胡伟武提出了主要思路, 撰写了论文整体框架; 汪文祥撰写了第 1 节部分内容; 吴瑞

Table 9 QEMU and LATX Performance of SPEC CPU2000

表 9 QEMU 和 LATX 翻译运行 SPEC CPU2000 的性能

SPEC 程序	QEMU 分值	LATX 分值	原生分值	$\frac{\text{QEMU}}{\text{原生}}$/%	$\frac{\text{LATX}}{\text{原生}}$/%	$\frac{\text{LATX}}{\text{QEMU}}$/%
164.gzip	522	1 122	1 332	39.2	84.23	214.9
175.vpr	259	1 454	2 351	11.0	61.85	561.4
176.gcc	605	1 982	3 120	19.4	63.53	327.6
181.mcf	1990	3 272	3 567	55.8	91.73	164.4
186.crafty	425	1 743	3 017	14.1	57.77	410.1
197.parser	492	1 537	2 103	23.4	73.09	312.4
252.eon	81	1 834	4 194	1.9	43.73	2 264.2
253.perlbmk	498	1 676	2 922	17.0	57.36	336.5
254.gap	497	1 686	2 512	19.8	67.12	339.2
255.vortex	537	2 083	3 550	15.1	58.68	387.9
256.bzip2	567	1 466	2 130	26.6	68.83	258.6
300.twolf	822	2 012	2 815	29.2	71.47	244.8
定点几何平均	485	1 763	2 692	18.0	65.50	363.5
168.wupwise	42	1 883	2 921	1.4	64.46	4 483.3
171.swim	83	3 462	5 637	1.5	61.42	4 171.1
172.mgrid	22	2 158	3 230	0.7	66.81	9 809.1
173.applu	42	2 255	2 482	1.7	90.85	5 369.0
177.mesa	104	1 514	3 451	3.0	43.87	1 455.8
178.galgel	70	6 739	6 552	1.1	102.85	9 627.1
179.art	187	6 336	8 962	2.1	70.70	3 388.2
183.equake	72	3 370	4 606	1.6	73.17	4 680.6
187.facerec	105	3 279	4 959	2.1	66.12	3 122.9
188.ammp	44	2 051	2 748	1.6	74.64	4 661.4
189.lucas	63	2 521	3 560	1.8	70.81	4 001.6
191.fma3d	58	2 427	3 431	1.7	70.74	4 184.5
200.sixtrack	10	905	1 433	0.7	63.15	9 095.0
301.apsi	57	2 948	3 569	1.6	82.60	5 171.9
浮点几何平均	56	2 630	3 740	1.5	70.30	4 697.0

阳和王焕东撰写了第 2 节部分内容;曾露撰写了第 3 节部分内容;徐成华、曾露和高翔撰写了第 4 节部分内容;张福新负责论文组织和统稿.

参 考 文 献

[1] Loongson Technology Corp. Ltd. LoongArch architecture manual[EB/OL]. [2022-06-28]. https://www.loongson.cn/loongArch

[2] Loongson Technology Corp. Ltd. Loong3A5000 processor[EB/OL]. [2022-06-28]. https://www.loongson.cn/productShow/32

[3] Bellard F. QEMU, a fast and portable dynamic translator[C] //Proc of the 2005 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2005: 41−47

[4] Loongson Technology Corp. Ltd. Loongnix-20. LoongArch64 [EB/OL]. [2021-03-31]. http://loongnix.cn

[5] Chernoff A, Herdeg M, Hookway R, et al. FX!32: A profile-directed binary translator[J]. IEEE Micro, 1998, 18(2): 56−64

[6] Baraz L, Devor T, Etzion O, et al. IA-32 execution layer: A two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems[C] //Proc of the 36th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2003: 191−201

[7] Dehnert J C, Grant B K, Banning J P, et al. The Transmeta Code Morphing(TM) software: Using speculation, recovery, and adaptive retranslation to address real-life challenges[C] //Proc of the Int Symp on Code Generation and Optimization: Feedback-directed and Runtime Optimization. Piscataway, NJ: IEEE, 2003: 15−24

[8] Ebcioglu K, Altman E. DAISY: Dynamic compilation for 100% architectural compatibility[C] //Proc of the 24th Annual Int Symp on Computer Architecture. New York: ACM, 1997: 26−37

[9] Moore R W, Baiocchi J A, Hisor J D, et al. Addressing the challenges of DBT for the ARM architecture [C] //Proc of the 2009 ACM SIGPLAN/SIGBED Conf on Languages, Compilers, and Tools for Embedded System. New York: ACM, 2009: 147−156

[10] Mittal A, Bansal A, Sethi V, et al. Efficient virtualization on embedded power architecture platforms[C] //Proc of the 8th Int Conf

on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2013: 445−458

[11] Ottoni G, Hartin T, Weaver C, et al. Harmonia: A transparent, efficient, and harmonious dynamic binary translator targeting the Intel architecture[C] //Proc of the 8th ACM Int Conf on Computing Frontiers. New York: ACM, 2011: 26:1−26:10

[12] Hu Weiwu, Wang Jian, Gao Xiang, et al. Godson-3: A scalable multicore RISC processor with x86 emulation[J]. IEEE Micro, 2009, 29(2): 17−29

[13] Hu Weiwu, Liu Qi, Wang Jian, et al. Efficient binary translation system with low hardware cost[C] //Proc of the IEEE Int Conf on Computer Design. Piscataway, NJ: IEEE, 2009: 305−312

[14] Hu Weiwu, Jin Guojie, Wang Wenxiang, et al. Fusion technology of Loongson instruction set architecture[J]. SCIENTIA SINICA Informationis, 2015, 45(4): 459−479 (in Chinese)
(胡伟武, 靳国杰, 汪文祥, 等, 龙芯指令系统融合技术[J]. 中国科学: 信息科学, 2015, 45(4): 459−479)

[15] Wine. Wine[EB/OL]. [2021-03-21]. https://www.winehq.org

[16] Spink T, Wagstaff H, Frank B. A retargetable system-level DBT hypervisor[C/OL] //Proc of the 2019 USENIX Annual Technical Conf. USENIX Association, 2019: 505−520 [2022-03-01]. https://www.usenix.org/conference/atc19/presentation/spink

[17] Huang Kele, Zhang Fuxin, Li Cun. BTMMU: An efficient and versatile cross-ISA memory virtualization[C] //Proc of the 17th ACM SIGPLAN/SIGOPS Int Conf on Virtual Execution Environments. New York: ACM, 2021: 71−83

[18] SPEC. SPEC CPU benchmarks[EB/OL]. [2021-03-21]. https://www.spec.org

[19] McCalpinJ.Streamv5.10[EB/OL].[2021-03-01].https://cs.virginia.edu/stream

[20] UnixBench. UnixBench v5.1.3[EB/OL]. [2021-03-21]. https://github.com/kdlucas/byte-unixbench/tree/master/UnixBench

Hu Weiwu, born in 1968. PhD, professor. Fellow of CCF. His main research interests include microprocessor design, computer architecture, and integrated circuit.

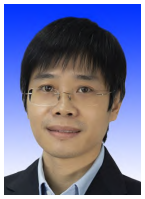
胡伟武, 1968 年生.博士，研究员.CCF 会士.主要研究方向为微处理器设计、计算机系统结构和集成电路.

Wang Wenxiang, born in 1983. PhD, senior engineer of Loongson Technology Corp. Ltd.. Member of CCF. His main research interests include high-performance computer architecture and processor microarchitecture.

汪文祥, 1983 年生.博士,龙芯中科技术股份有限公司资深工程师.CCF 会员.主要研究方向为高性能计算机体系结构和处理器微体系结构.

Wu Ruiyang, born in 1991. PhD. His main research insterest is computer architecture.

吴瑞阳, 1991 年生.博士.主要研究方向为计算机体系结构.



Wang Huandong, born in 1982. PhD, chief engineer of Loongson Technology Corp. Ltd., assistant professor. Member of CCF. His main research interests include multi-core processor architecture, memory interface design and high-speed I/O interface design.

王焕东, 1982 年生.博士,龙芯中科技术股份有限公司首席工程师，助理研究员.CCF 会员.主要研究方向为多核处理器体系结构、内存接口设计和高速 I/O 设计.

Zeng Lu, born in 1987. PhD, engineer. His main research interests include binary translation and computer architecture.

曾　露, 1987 年生.博士，工程师.主要研究方向为二进制翻译与计算机体系结构.

Xu Chenghua, born in 1989. PhD candidate. His main research interests include performance analysis, compiler design and optimization.

徐成华, 1989 年生.博士研究生.主要研究方向为性能分析、编译器设计与优化.

Gao Xiang, born in 1982. PhD, professor. Member of CCF. His research interests include high-performance computer architecture, parallel processing, and operating systems.

高　翔, 1982 年生.博士，正高级工程师.CCF 会员.主要研究方向为高性能计算机体系结构、并行处理和操作系统.

Zhang Fuxin, born in 1976. PhD, professor. Senior member of CCF. His main research interests include computer architecture, binary translation, and operation system. (fxzhang@ict.ac.cn)

张福新, 1976 年生.博士，正高级工程师.CCF 高级会员.主要研究方向为计算机体系结构、二进制翻译和操作系统.