第 35 卷 第 7 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.7 Computer Engineering April 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)07—0280—03 文献标识码：A 中图分类号：TP311.52

基于龙芯处理器的二进制翻译器优化

蔡嵩松[1,2]，刘 奇[2]，王 剑[2]，刘金刚[1]
(1. 首都师范大学和中国科学院计算技术研究所计算机科学联合研究院，北京 100037；2. 中国科学院计算技术研究所，北京 100080)

摘 要：二进制翻译是实现系统迁移的主要方法，但基于通用平台的仅靠软件实现的二进制翻译性能不高。该文以龙芯 2F 处理器为实现平台，提出一种 QEMU 二进制翻译器并进行优化，其中包括编译环境的优化以及二进制翻译器本身的优化 2 个方面，对后者的优化主要涉及寄存器直接映射和多媒体指令的改进。实验结果表明，通过寄存器映射优化后，系统能够获得 1.45 的加速比，通过多媒体优化后，多媒体程序的执行能达到本地机器执行的 80%的性能。

关键词：龙芯 2F 处理器；二进制翻译器；寄存器；堆栈

Optimization of Binary Translator Based on GODSON CPU

CAI Song-song[1,2], LIU Qi[2], WANG Jian[2], LIU Jin-gang[1]
(1. Joint Academe of Computer Science, Capital Normal University & Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100037; 2. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

【Abstract】Binary translation is the most important approach to achieve system migration, but the one based on general architectural and software suffers from poor performance. This paper introduces an optimization method of QEMU binary translation based on GODSON2F CPU. This optimization mainly contains two aspects: tool chain and QEMU binary translator itself. Aiming at the latter one, the methods are direct mapping of registers and the optimization to Multimedia Extensions(MMX) instructions. Experimental results show that the OS obtains speed-up of 1.45 through direct mapping of registers, and the MMX programs attains 80% performance of the host machine through the optimization to MMX instructions.

【Key words】GODSON2F CPU; binary translator; register; stack

1 概述

目前 X86 处理器在应用中占据主导地位，这是因为在服务器领域中，大量应用都是 X86 架构，RISC 架构的微处理器为了能够广泛地运行服务器类的应用，实现与 X86 架构的兼容就成为系统实现的首要任务，另外，X86 架构的桌面应用程序更加多样化，很多商业软件都基于 X86 架构的，所以 RISC 微处理器要想更广泛地运行多样化的桌面应用，就需要实现对 X86 的兼容，而二进制翻译是实现系统迁移的重要方法，因此，二进制翻译器的应用也就越来越广泛。

虚拟机主要分为 2 类：进程级虚拟机和系统级虚拟机。进程级虚拟机提供 ABI 级的支持，包括用户指令及运行环境(USR)和系统调用(SYSCALL)；系统级虚拟机支持完整的 ISA，包括用户指令及运行环境(USR)和特权指令及运行环境(PRV)。

实现不同系统的迁移是模拟器的工作[1-2]，模拟器的实现有 2 种方法：解释执行和翻译执行。解释执行就是单条指令的解释，每次解释一条目标指令代码为本地指令代码，然后执行这条指令，这样一条一条指令地执行完所有程序；翻译执行是按照指令块进行翻译，翻译好的指令块存入 Translation Block(TB)中，以后每执行到这段代码时就到 TB 中取指令，从而省去二次翻译的过程，在指令多次重复执行的情况下，效率相对于单条指令的解释执行有较大提升。

QEMU 是个二进制动态翻译器[3]，它有系统级和进程级 2 种工作模式，且可以实现多种异构平台的映射。具有依赖平台多样、翻译效率相对较高、开源易移植等优点，因此，选择 QEMU 作为二进制翻译的主要媒介，并在此基础上针对龙芯 2F 处理器作相应优化，以达到龙芯处理器兼容 X86 应用程序的目的。

2 QEMU到龙芯的移植

QEMU 开源代码中有支持 X86 架构的稳定版本，由于龙芯 2F 是基于 MIPS 架构的，而从 X86 到 MIPS 程序在控制行为上最主要的差别就是程序的调用方式，因此将这个版本移植到龙芯 2F 上的主要工作是消除它们程序调用方式的差异。

X86 程序在函数调用时，返回地址保存在堆栈中，函数返回时从返回地址出栈即可；而 MIPS 程序在函数调用时，将返回地址保存在 31 号寄存器中，返回时由跳转指令跳转到

基金项目：国家“973”计划基金资助项目“可扩展、可重构片上并行体系结构与原型芯片设计”(2005CB321601)；国家“863”计划基金资助重点项目“低成本先进计算机单机”(2006AA010201)；国家“863”计划基金资助重点项目“大规模片上多处理器高性能存储系统研究”(2007AA01Z114)；国家自然科学基金资助项目“共享二级 Cache 的片上多处理器 Cache 块分布技术研究”(60703017)；国家自然科学基金资助项目“高性能片上存储系统”(60736012)；国家自然科学基金资助项目“资源有效的单片多处理器结构研究”(60673146)；北京市自然科学基金资助项目“一种创新的同时多个微线程处理器关键技术研究”(4072024)

作者简介：蔡嵩松(1986－)，男，硕士研究生，主研方向：计算机体系结构；刘 奇，博士研究生；王 剑，副研究员；刘金刚，教授、博士生导师

收稿日期：2008-08-30 **E-mail：**caisongsong@ict.ac.cn

31 号寄存器保存的地址。MIPS 的这种函数调用的方式会引发一系列问题：二进制翻译器上层的程序在函数调用时，返回地址使用 31 号寄存器，这就会破坏 QEMU 本身的状态，QEMU 的函数调用返回时就会出现错误。

因此，本文在二进制翻译器上运行的程序调用点做标记，每出现这样的调用，返回地址保存在内存中，而不是 31 号寄存器中，返回时从内存中返回，这就可以保持 QEMU 本身的状态。

3 编译环境的优化

在对 QEMU 结构优化之前，先针对龙芯 2F 的结构[4-5]优化编译环境，使得编译环境在编译 QEMU 时能够编译出针对龙芯 2F 处理器更有效率的 QEMU 代码，这也是提高 QEMU 性能的一个重要方面。

龙芯 2F 使用 GNU binutils, GCC 和 GLIBC 作为基本的编译环境。

对于 GNU binutils 汇编器，在汇编器的 gas 中加入龙芯 2F 自定义的多媒体指令，从而手工编写的多媒体指令就可以由编译器编译成目标代码，为 QEMU 多媒体指令的优化做准备；对于 GCC，根据龙芯 2F 处理器中指令的特点，在 GCC 中加入新的指令模板和指令流水线描述，使 GCC 编译出的汇编指令更适合于龙芯 2F 处理器。

对于 GLIBC 库，通过 profile，统计 GLIBC 中使用次数频繁的函数，根据龙芯 2F 的结构重新改写这些函数。

为充分利用龙芯 2F 处理器 64 bit 的资源，编译工具链使用 N32 的 ABI 取代原来的 O32 的 ABI 工具链。O32 的工具链编译的目标程序把处理器看作 32 bit，而 N32 的工具链编译的目标程序把处理器看作 64 bit，虽然 O32 的工具链具有兼容性好的优点，但在资源利用方面显然后者更出色，更有助于发掘出处理器最大的性能。

经过上述对编译环境的优化，编译 QEMU 得到的目标代码性能会有大幅度提高。

4 QEMU二进制翻译器结构优化

4.1 寄存器分配优化

QEMU 采用固定寄存器分配方法，这种方法将目标机的寄存器固定地映射到本机相应的寄存器或者内存中的某个地址。QEMU 对于大部分的本地机器，它只是将临时变量保存在本地机器的 3 个寄存器中，其他大部分的目标寄存器映射到内存中，临时变量对于不同的本地机器分配到不同的寄存器中。这种方法的优点是简便并且可移植性好，但是这样做的性能开销太大，由于在本机是用内存模拟的目标寄存器，当用到目标寄存器时，就会出现大量的访存操作，因此势必会降低二进制翻译器的效率。一种可行的做法是对于某种特定结构，尽可能地将目标机程序最常用的寄存器固定映射到本地机器地寄存器中，这样寄存器操作时的多余访存就可以避免，这是种牺牲二进制翻译器的灵活性而达到提高性能目的的方法。

针对龙芯 2F 处理器优化 QEMU，让 QEMU 局限于该结构即可，从而可以将 X86 目标机的寄存器固定一对一映射于龙芯 2F 的寄存器上。

X86 共有 8 个通用寄存器，分别为 EAX, EBX, ECX, EDX, ESI, EDI, ESP 和 EBP。通过对 QEMU 二进制翻译器上 Windows98 操作系统启动过程的 profile，得到 X86 架构下操作系统启动过程对这 8 个通用寄存器的访问次数及其所占百分比，如表 1 所示。

表 1 X86 架构下 OS 启动过程对寄存器的访问频度

通用寄存器	访问次数	访问比例/(%)
EAX	258 379	26.7
EBX	94 118	9.7
ECX	76 517	7.9
EDX	109 305	11.3
EBP	125 590	13.0
ESI	80 132	8.3
EDI	83 493	8.6
ESP	139 802	14.5

由上述结果可以得到操作系统在 X86 架构运行时最常用的通用寄存器是 EAX, ESP, EBP，所以，将这 3 个最常用的寄存器映射于 s4, s5 和 s6 寄存器，另外定义 s0, s1, s2 和 s3 4 个寄存器保存临时变量，并且修改编译器，使得其他程序尽量不使用这 7 个寄存器，如果其他程序使用这些寄存器，需要首先把这些寄存器中的值保存到内存中，程序退出后再将这些寄存器的值从内存中恢复回来。

对于寄存器关联的这种优化，可以使运行在 QEMU 上的 X86 系统或程序在使用 EAX, ESP, EBP 这 3 个寄存器时，省去不必要的访存操作，从而提高目标机寄存器操作的效率。

4.2 多媒体处理优化

多媒体技术是 Intel 公司在处理器中为增强处理器的媒体处理能力而引入的一项技术，分为 MMX, SSE, SSE2 和 SSE3 4 类。MMX 技术是 Intel 公司在奔腾Ⅱ的 IA-32 架构中引入的，它实现了单指令流多数据流(SIMD)，增强了媒体处理的性能。

龙芯 2F 处理器中定义自己的多媒体指令，这些多媒体指令一对一实现 MMX 指令的所有功能，因此，可以使用龙芯 2F 处理器中的这些多媒体指令对 QEMU 二进制翻译器中的多媒体处理部分进行优化。

QEMU 中对于多媒体处理部分是软件模拟，每条 X86 的多媒体指令都是用一个相应的函数来实现，一条指令经过翻译后会变成很多条指令，这会影响二进制翻译器的效率。这种做法是通用的做法，扩展性好，可以适用于 X86 架构到其他多种架构的翻译，但是它是以牺牲性能为代价的。

由于龙芯 2F 处理器中定义了跟 MMX 指令相似的多媒体指令，因此可以对 QEMU 进行优化，把 QEMU 中多媒体指令用相应的龙芯指令来实现，而放弃原有的函数实现，这样一条 X86 的 MMX 指令就可以翻译成为一条对应的龙芯指令了，从而二进制翻译器的性能可以大幅提高，翻译出来的代码的性能几乎可以达到本地机器上执行相应代码的性能。这种做法使 QEMU 固定于特定的龙芯处理器，但是二进制翻译器的媒体处理部分性能可以得到大幅度的提高。

例如，一条 x86 的多媒体指令：

```
punpcklwd  mm0,  mm1
```

该指令实现将 MMX 寄存器 0 和 1 中的数据按一定顺序进行拼接，结果存放在 MMX 寄存器 0 中。

优化前，二进制翻译器仅靠软件的方法翻译为以下一段代码：

```
sw    $0,244($19)
sw    $0,256($19)
sw    $0,260($19)
lui   $6,0x0
ori   $6,$6,0x108
```

```
addu   $6,$6,$19
lhu    $2,0($6)
addiu  $10,$10,-16
sh     $2,8($10)
lui    $5,0x0
ori    $5,$5,0x110
addu   $5,$5,$19
lhu    $2,0($5)
nop
sh     $2,10($10)
lhu    $3,2($6)
lw     $2,8($10)
sh     $3,12($10)
lhu    $4,2($5)
nop
sh     $4,14($10)
lw     $3,12($10)
addiu  $10,$10,16
sw     $2,0($6)
sw     $3,4($6)
```

由于仅靠软件翻译，因此一条 X86 指令翻译后会产生 25 条 MIPS 代码，翻译效率比较低。

优化后，二进制翻译器翻译得到如下指令，该指令是与原 X86 指令一一对应的龙芯 2F 的多媒体指令。

```
gspunpcklwd   $f20, $f20, $f22
```

5 测试结果

5.1 测试环境

本文在龙芯 2F 处理器上进行数据测试。龙芯 2F 处理器是款 64 bit、四发射、乱序执行的 RISC 处理器，实现 MIPS Ⅲ指令集。该处理器采用先进的乱序执行技术(如寄存器重命名、转移预测和动态调度等)和 Cache 技术(如非阻塞 Cache、load 猜测、动态内存相关和写合并技术等)，并集成片上二级 Cache、DDR2 内存控制器和 I/O 控制器来提高流水线效率及 I/O 能力。

5.2 结果分析

5.2.1 寄存器优化测试结果

由于通用寄存器影响的主要是系统进程切换时的性能，为测试寄存器映射部分的优化效果，选择使用在 QEMU 上操作系统启动和关机时间来对比优化前后的二进制翻译器的效率。在二进制翻译器上运行 Windows98 操作系统，并且设置系统启动后马上自动关闭，计算系统从开始启动到关闭的时间，测试结果如表 2 所示。

表 2 寄存器优化部分测试结果

测试程序	优化前/s	优化后/s	加速比
操作系统开关机时间	37.759	26.018	1.45

经过优化后，系统在使用目标寄存器 EAX, EBP 和 ESP 时不需要进行访存操作，而是直接使用龙芯 2F 的 s4, s5 和 s6 寄存器，因此，减少了系统的启动关闭时间，获得较高的加速比。

5.2.2 多媒体部分优化测试结果

为测试 QEMU 多媒体部分优化的效果，选择 3 个典型应用的 MMX 算法加以实现，分别是图像压缩中的 IDCT 算法、快速傅里叶算法以及像素转换的 YUV2RGB 算法，这 3 个算法用 X86 MMX 汇编指令手工实现，将程序用优化过的 QEMU 进行翻译，翻译过的代码提取后在龙芯 2F 处理器上运行。把这个性能同没有优化过的 QEMU 上翻译的代码进行比较，从而计算加速比。另外，将这 3 个算法用 MIPS MMX 汇编指令在龙芯 2F 上手工实现，得到本机执行的理想时间。定义优化过的翻译代码的执行时间和本机执行的理想时间的比率为二进制翻译器的效率，各种测试结果如表 3 所示。

表 3 多媒体优化部分测试结果

测试程序	优化前/时钟周期数	优化后/时钟周期数	理想情况/时钟周期数	加速比	二进制翻译器效率/(%)
IDCT 算法	87 262 827	5 544 017	4 903 129	15.74	88.44
FFT 算法	77 309 358	3 240 124	2 968 926	23.86	91.63
YUV2RGB 算法	9 628 120	350 879	313 405	27.44	89.32

经过优化后，这几个程序翻译得到的代码接近于一条 MIPS 指令对应于一条 X86 指令，所以，优化后的结果很接近于理想情况，这就是优化后二进制翻译器效率较高的原因。优化前的二进制翻译器仅仅利用软件翻译，一条 X86 指令翻译出的代码往往对应于十几条甚至几十条 MIPS 指令，导致效率的低下，这也是优化后加速比较高的原因。

6 结束语

由于 QEMU 二进制翻译器的内部结构本身支持多平台，翻译结构采用软件模拟，因此一条目标指令翻译后通常会变成几十条本地指令，另外，对于目标机各种结构的处理也是软件模拟，这会导致翻译效率低下。本文针对龙芯 2F 处理器的特定结构，从各方面描述 QEMU 二进制翻译器针对龙芯 2F 处理器特定结构的优化。

在系统方面，优化编译器环境，使编译出的 QEMU 代码更适合于龙芯 2F 的结构，在 QEMU 本身方面，进行寄存器映射和 MMX 指令 2 方面的优化，虽然使 QEMU 仅限于龙芯 2F 处理器，但提高了翻译效率。

参考文献

[1] Anton C, Mark H. FX!32: A Profile-directed Binary Translator[J]. IEEE Micro, 1998, 18(2): 56-64.

[2] Leonid B, Tevi D. 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'l Symp. on Microarchitecture. [S. l.]: IEEE Press, 2003.

[3] Fabrice B. QEMU, A Fast and Portable Dynamic Translator[C]//Proc. of the 2005 USENIX Annual Technical Conference. Berlin, Germany: Springer-Verlag, 2005.

[4] Hu Weiwu, Zhang Fuxin, Li Zusong. Microarchitecture of the Goodson-2 Processor[J]. Journal of Computer Science and Technology, 2005, 20(2): 243-249.

[5] Hu Weiwu, Tang Zhimin. Microarchitecture Design of the Godson-1 Processor(in Chinese)[J]. Chinese Journal of Computers, 2003, 26(4): 385-396.

编辑 陈 文