计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140220
52(9):2105-2113,2015

基于高速缓存负荷均衡的动态二进制翻译研究

李战辉[1] 刘 畅[1] 孟建熠[2] 严晓浪[1,2]
[1](浙江大学超大规模集成电路设计研究所 杭州 310027)
[2](专用集成电路与系统国家重点实验室(复旦大学) 上海 201203)
(lizh2016@zju.edu.cn)

Cache Load Balancing Oriented Dynamic Binary Translation

Li Zhanhui[1], Liu Chang[1], Meng Jianyi[2], and Yan Xiaolang[1,2]
[1](*Institute of Very Large Scale Integrated Circuits Design, Zhejiang University, Hangzhou* 310027)
[2](*State Key Laboratory of Application Specific Integrated Circuit & System(Fudan University), Shanghai* 201203)

Abstract Based on the fact that the highly increasing load of instruction cache and data cache has led to great performance loss for DBT (dynamic binary translator), and the out-of-balance increasing rate between instruction cache and data cache makes the situation worse, this paper proposes a hardware-software-codesigned DBT acceleration mechanism that speeds up DBT performance by dynamically balancing load of instruction cache to data cache. The key idea of this mechanism is the design of the cache load balancing state for microprocessors. When microprocessor working in this state, the instruction cache stays the same as usual and the data cache is divided into two areas: normal-accessing-area and load-balancing-area. The normal-accessing-area caches regular program data just as the traditional data cache does. However, the load-balancing-area is quite different. It doesn't cache regular program data, but supports load-transforming-channel, which is used to transform and assimilate most of the instruction cache load caused by scheduler of the DBT. Main work of the scheduler is converting jump target address from source machine code space to target machine code space. Experimental results based on EEMBC (embedded microprocessor benchmark consortium) benchmarks show that the access load of instruction cache is reduced by 35%, data cache is reduced by 58%, and overall performance of the QEMU(quick emulator) DBT is improved by 171%.

Key words dynamic binary translation; indirect jump translation; cache load; load balancing area (LBA); load transforming channel (LTC)

摘 要 针对动态翻译时指令和数据高速缓存访问负荷大幅增加且增幅不均衡导致翻译器性能下降的问题,提出基于指令高速缓存与数据高速缓存访问负荷动态均衡的软硬件协同翻译方法.该方法为处理器设计高速缓存负荷平衡状态,该状态将数据高速缓存分为普通区和负荷平衡区(load balancing area, LBA),普通区缓存正常的程序数据,负荷平衡区通过负荷转化通道(load transforming channel, LTC)吸收动态翻译器调度器地址空间转换操作在指令高速缓存上产生的部分负荷,以提高数据高速缓存利用率.EEMBC(embedded microprocessor benchmark consortium)测试基准实验结果表明,在同等处理器资源的情况下,该方法将指令高速缓存访问次数平均减少35%,数据高速缓存访问次数平均减少58%,动态翻译器综合性能提高171%.

收稿日期:2014-03-13;**修回日期**:2014-11-20
基金项目:中央高校基本科研业务费专项基金项目(2012QNA5004)
通信作者:孟建熠(mengjy@fudan.edu.cn)

关键词 动态二进制翻译;间接转移翻译;高速缓存负荷;负荷平衡区;负荷转换通道

中图法分类号 TP332; TP314

动态翻译[1-2]将针对源体系架构(源机器)编译生成的二进制代码(源程序)翻译为可以在目标体系架构(目标机)上运行的代码(翻译码),然后动态执行.动态翻译器(dynamic binary translator, DBT)主要包括映射器、调度器和执行器3个模块.映射器把源程序载入数据高速缓存(data cache, DCache)并映射为翻译码,并把翻译码存储到内存和DCache;执行器执行翻译码并把翻译码载入指令高速缓存(instruction cache, ICache);调度器完成翻译码异常处理、映射器和执行器之间的切换以及源机器到目标机地址空间转换操作.相比源机器执行源程序,动态翻译时高速缓存负荷数倍膨胀,膨胀主要来自:1)映射器把源程序映射为翻译码时的缓存负荷;2)调度器调度操作导致的缓存负荷;3)执行器对相比源程序有数倍代码膨胀比的翻译码的执行导致的缓存负荷.高速缓存负荷严重膨胀是动态翻译器性能下降的主要原因之一.

针对执行器高速缓存负荷优化:文献[3-5]通过不同的寄存器分配窗口和分配算法,文献[6-8]通过删除冗余标志位计算和延迟标志位计算等方法减少生成的翻译码指令数,以优化缓存访问负荷;文献[9]采用轻量级的寄存器生命周期分析算法优化翻译码,以减少翻译码指令数;文献[10-12]在轻量级优化的基础上对热点路径翻译码进行再组织,以进一步减少翻译码指令数.调度器高速缓存负荷优化针对间接转移指令:文献[13-14]在文献[8]软件Hash表的基础上提出转移目标地址预测法,预测成功时优于Hash表法,失败时再次查找Hash表缓存负荷更大;文献[15]将间接转移指令翻译码存储在源机器代码空间对应的跳转目标地址处,无需地址转换操作,缓存负荷较小,但是当转移指令密集出现时,复杂的源程序保护机制将导致更大的缓存负荷;文献[16-17]给处理器增加内容可寻址存储器,解决间接转移指令地址转换操作,缓存负荷很低,但是需引入额外硬件资源并进行全定制设计.映射器高速缓存负荷主要来自对源程序的翻译,动态翻译器通过存储翻译码来避免重复翻译导致的缓存负荷,但是当翻译码存储空间不足时,频繁刷新该空间带来的重复翻译会导致较大缓存负荷:文献[18-19]通过精简翻译码代码量以尽量减少因为存储空间不足导致的重复翻译带来的高速缓存负荷;文献[20-21]将翻译码存储空间分区,通过优先刷新不常使用的分区来减少重复翻译导致的高速缓存负荷,当存储空间充足时,这些方法引入的额外操作反倒给动态翻译器带来更大的缓存负荷;文献[16-22]通过向目标处理器引入X86的硬件译码器来加速代码转译过程,缓存负荷很小,但需引入额外硬件资源.

本文以QEMU(quick emulator)[8]动态翻译框架分析动态翻译器对一级高速缓存的访问特性,发现高速缓存数据量和访问负荷提高明显且存在ICache和DCache不均衡的情况,这是动态翻译器的主要性能瓶颈,进而提出动态均衡指令高速缓存与数据高速缓存负荷以平衡高速缓存利用率的软硬件协同翻译方法,主要内容与创新点包括以下3点:

1) 研究动态翻译器高速缓存性能表现,发现:

① 动态翻译器高速缓存数据量和访问负荷增加且不均衡.ICache和DCache访问次数分别增加7.0倍和5.1倍,ICache数据量是DCache的1.3倍,ICache数据量大且被频繁使用,DCache数据量较小且一半左右为单次使用数据.

② 调度器是高速缓存负荷增加的主因之一,其缓存访问次数分别占ICache和DCache总访问次数的27%和36%.

2) 提出动态均衡指令高速缓存与数据高速缓存负荷以平衡高速缓存利用率的动态翻译方法,该方法为处理器设计高速缓存负荷平衡状态和负荷转化通道,通过软硬件协同配合的方式把调度器地址转换操作在ICache上产生的负荷转化到DCache,有效提高DCache利用率和动态翻译器性能.

3) 实验分析本方法运行基准测试程序EEMBC(embedded microprocessor benchmark consortium)时的性能,在同等处理器资源的情况下,本方法将ICache访问次数减少35%、DCache访问次数减少58%、动态翻译器综合性能提高171%.

1 高速缓存负荷分析

动态翻译器以基本翻译块为基本翻译单位,一般以转移指令作为基本翻译块的结束条件,典型工作流程如图1所示.图1中SPC(source machine program counter)指源机器程序计数器,TPC(target machine program counter)指目标机程序计数器.

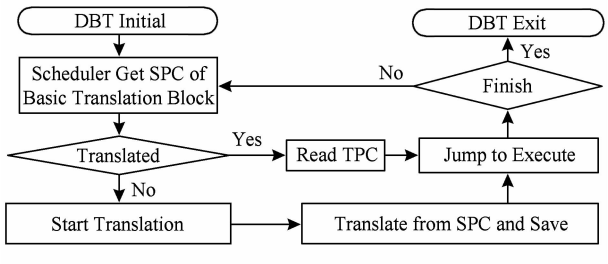


Fig. 1 Work flow of DBT.

图 1 DBT 工作流程

1.1 动态翻译器性能概述

首先向 QEMU 加入对国产自主嵌入式处理器 CK810[23] 系统架构的支持,将 ARM 系统架构转译至 CSKY 系统架构. 图 2 分析 CK810 通过 QEMU 运行 Cortex-A9 编译器编译出的 EEMBC 基准测试程序,相比直接运行 CK810 编译器编译出的测试程序性能,观察发现动态翻译性能不足原来的 20%.

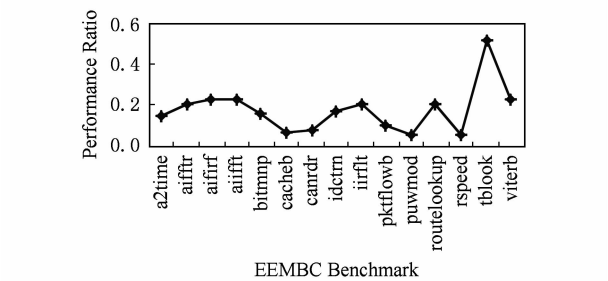


Fig. 2 Performance ratio of DBT than native execution.

图 2 DBT 性能相比本地执行性能百分比

1.2 高速缓存特性分析

动态翻译器对高速缓存的利用有着较严重的应用特征,本文重点研究二进制转译时处理器对高速缓存的调度方式,并通过软硬件协同配合的方法提高动态翻译器性能.

1) 高速缓存数据特性分析

动态翻译时, ICache 主要缓存翻译码以及执行器和映射器代码,在不考虑执行器和映射器代码对 ICache 影响的前提下,翻译码已经使得动态翻译器 ICache 数据量相比源程序数倍膨胀;DCache 主要缓存翻译码数据、映射器数据、调度器数据和初始化数据,翻译码数据对应于源程序数据,其他数据则是动态翻译器相比源程序给 DCache 额外增加的数据,其中映射器数据包括源程序、翻译上下文、中间微指令和翻译码等,调度器数据包括地址转换 Hash 表、源机器状态和目标机状态等. 对于 ICache,这些增加的数据被频繁访问,ICache 利用率很高;但是对于 DCache,很多增加的数据只被使用 1 次(如源程序在翻译时被载入 DCache,翻译后不再使用;翻译码先被存储在 DCache,然后同步到 ICache;初始化数据只用于初始化操作),这些单次使用数据(one usage data, OUD)不但无法利用 DCache 加速访问,反而可能将 DCache 有效数据替换出去,降低性能. 图 3 分析动态翻译器 ICache 和 OUD 数据量相比 DCache 数据量的归一化比例, 观察发现 ICache 数据访问量是 DCache 的 1.3 倍,而且 DCache 一半左右数据为单次使用数据.

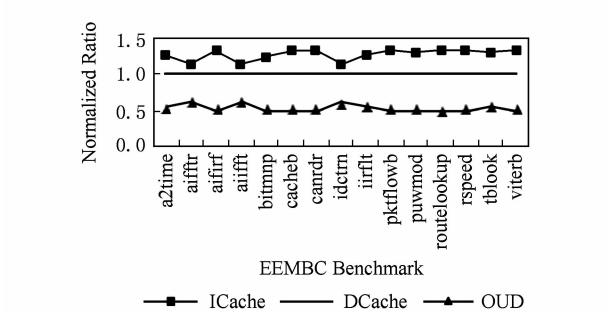


Fig. 3 ICache and OUD data size vs DCache data size.

图 3 ICache 和 OUD 的数据量相比 DCache 的比例

2) 高速缓存访问特性分析

图 4 分析动态翻译器相比源程序直接执行时高速缓存访问负荷的增加比例. 观察发现所有程序的高速缓存访问负荷都有大幅增加,其中 ICache 访问次数平均增加 7.0 倍,DCache 增加 5.1 倍. 图 5 进一步分析高速缓存访问负荷来源,观察发现 ICache 和 DCache 的访问负荷都来自执行器和调度器,执行器高速缓存访问负荷来自相比源程序有数倍代码膨胀比的翻译码的执行,本文重点针对调度器地址空间转换操作进行优化,以平衡高速缓存利用率.

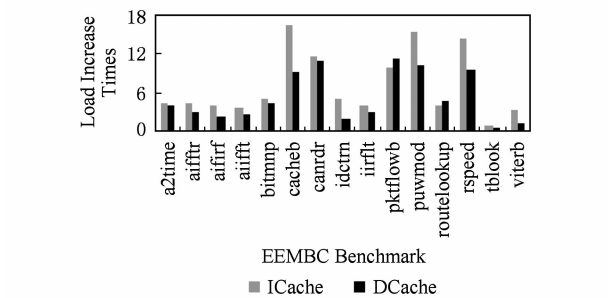


Fig. 4 Cache access time increase times than native execution.

图 4 DBT 缓存访问次数增加倍数

调度器调度操作包括翻译码异常处理、映射器和执行器互相切换以及源机器到目标机地址空间转换. 前两者较少出现;后者是高速缓存访问负荷的主要来源,用于解决源程序到翻译码地址的重定位问题,

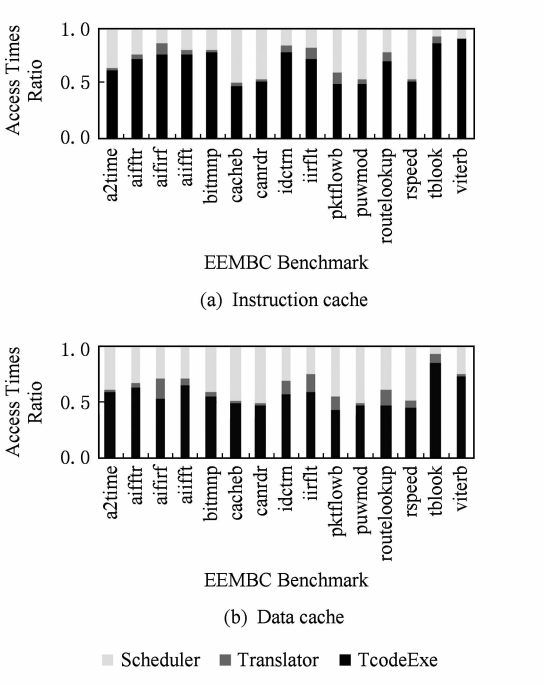


Fig. 5 Distribution of cache access times.

图 5 缓存访问次数分布

主要针对转移指令. 直接转移指令采用静态链接技术[7]避免该转换操作,间接转移指令转移的目标地址来自通用寄存器,具有不定性和多向性,其地址转换操作是动态翻译领域的研究热点[24].

QEMU 的软件 Hash 表地址转换法 ICache 负荷主要来自 Hash 表访问指令,每次访问累计至少需要 14 条指令(如表 1 所示),这些指令需要 56 B 的 ICache 容量;DCache 负荷主要来自 Hash 表数据,以 EEMBC 基准测试程序为例,每个程序平均有 41 个跳转次数大于 15 的热点间接转移目标地址,

Table 1 Access Flow for Hash Table

表 1 Hash 表访问流程

Number	Operation	Instruction Number
1	Argument prepare and call Hash function	2
2	Save registers	>1
3	Calculate Hash offset	1
4	Get Hash table address	1
5	Get Hash pointer	1
6	Null pointer check	2
7	Get source machine jump address if not null	1
8	Check hit or not	2
9	Get target machine jump address if hit	1
10	Restore registers	>1
11	Return	1

这些地址需要 41 个 Hash 表项,每个表项 12 B(表项地址、源机器地址和目标机地址),因此至少需要 492 B DCache 容量. 调度器对 ICache 和 DCache 访问次数相当,其热点程序对 ICache 容量需求却远低于 DCache,调度器对 ICache 利用率高于 DCache.

综合以上实验发现,动态翻译器高速缓存数据量以及访问负荷严重膨胀,这种膨胀造成 ICache 数据量大且被频繁使用,DCache 数据量较小且一半为单次使用数据,ICache 利用率远高于 DCache. 调度器对动态翻译器 ICache 和 DCache 访问负荷的增加起着关键作用,而且进一步增大了高速缓存利用率失衡的程度.

1.3 高速缓存利用率优化

优化思路是利用 DCache 分担部分 ICache 负荷以平衡高速缓存利用率. 如果直接使用 DCache 作为 ICache,则只会改善 ICache 缺失率,图 6 显示高速缓存缺失率平均不足 1%,优化缺失率并不会对性能带来大的提高. 本文利用部分专有 DCache 压缩并吸收调度器地址空间转换操作对 ICache 的访问负荷,以大幅提高动态翻译器综合效率,压缩基于高速缓存访问和 Hash 表查找的相似性.

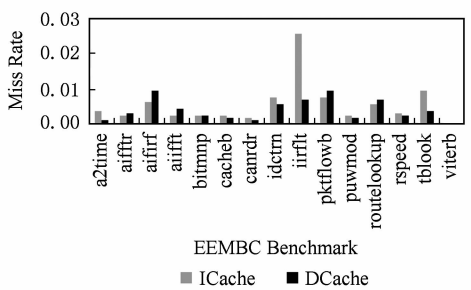


Fig. 6 Cache miss rate.

图 6 高速缓存缺失率

综上所述,本文提出动态均衡 ICache 与 DCache 访问负荷,以平衡高速缓存利用率的动态翻译方法,该方法采用软硬件协同的方式复用部分 DCache 压缩,并吸收调度器对 ICache 的负荷,以提高 DCache 利用率并加速动态翻译器性能. 本方法还可用于加速非动态翻译程序执行时的查找操作.

2 高速缓存负荷平衡算法

采用软硬件协同设计的方式实现本算法,硬件方面为处理器新增高速缓存负荷平衡状态,该状态提供 ICache 到 DCache 负荷转化通道(load transforming channel, LTC);软件方面修改映射器对间

接转移指令的处理,以利用负荷转化通道完成负荷平衡操作,加速调度器地址空间转换操作,提高动态翻译器性能.引入本方法后的调度器地址空间转换操作如图7所示:

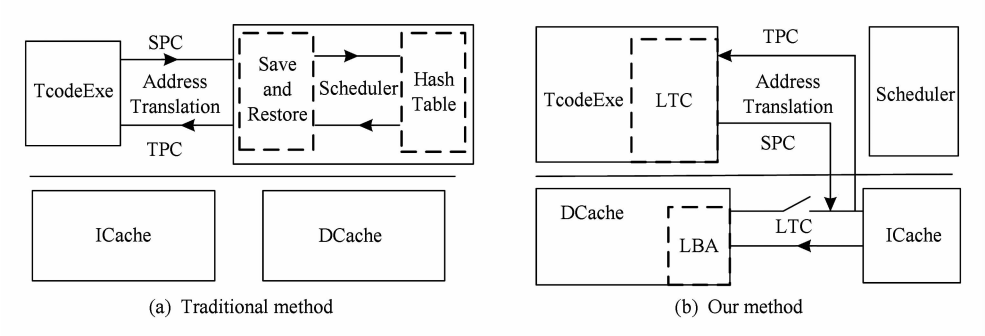


Fig. 7 Address translation of scheduler.

图7 调度器地址转换操作

2.1 硬件结构设计

硬件结构设计主要为处理器新增高速缓存负荷平衡状态和负荷转化通道.负荷平衡状态将DCache分为普通区和负荷平衡区(load balancing area,LBA),普通区缓存正常的程序数据,负荷平衡区提供负荷转化通道,以利用部分DCache压缩并吸收调度器地址空间转换操作对ICache的负荷.

1) 负荷平衡区

设计的关键是在较小硬件代价和设计复杂度下,划分出部分DCache作为负荷平衡区.本文复用CK810处理器4路组相连DCache第4路末尾2KB作为负荷平衡区,负荷平衡区同样采用4路组相连结构.如图8所示,将DCache第4路的数据存储器分为5块,第1块6KB,其余4块各0.5KB.处理器工作在高速缓存正常执行状态时,这5块高速缓存组合为1块8KB的高速缓存来处理正常的程序数据;处理器工作在负荷平衡状态时,第1块作为普通区缓存正常的程序数据,其余4块作为负荷平衡区处理地址转换操作.当数据加载存储指令的地址索引到每路的末尾2KB且缓存工作在负荷平衡状态时,这些指令只可读写前3路DCache,第4路DCache对于这类指令缺失;否则,这些指令可读写全部4路DCache.要将1路的DCache扩展到4路,需引入额外存储资源来缓存这4路负荷平衡区的有效位和标记位.本文采用如下方案避免上述存储资源:CK810 DCache每个数据块为32B,用32B中的16B存储标记位和有效位,另外16B存储数据.

上述负荷平衡区实现机制,只需对处理器原有的DCache控制逻辑进行小的修改即可实现,几乎无需增加额外硬件资源.同时,本方法只影响地址刚好索引到DCache每路末尾2KB、且DCache第1,2,3路已满时的数据加载存储指令的性能,对大多数数据加载存储指令性能无影响.

2) 负荷转化通道

负荷转化通道完成对负荷平衡区的读写,以实现调度器ICache负荷到DCache的转化,通道操作包括激活和关闭以及负荷压缩和转化.通道激活和关闭通过切换高速缓存工作状态实现,只需在原有处理器状态寄存器中新增缓存工作状态指示位即可.负荷压缩和转换利用高速缓存访问和Hash表

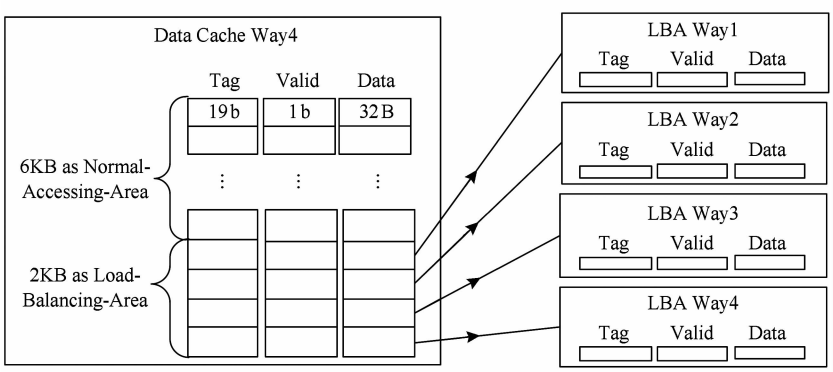


Fig. 8 Data cache structure for LBA.

图8 负荷平衡区高速缓存设计图

查找的相似性,将调度器地址转换时的 Hash 表查找过程等效为 1 次负荷平衡区访问操作,通过负荷转换通道完成,需增加负荷转换通道读写指令实现.

定义 1. 负荷转换通道读写指令(LTC. R/W). LTC. R 以源机器代码空间地址为索引,通过负荷转化通道读负荷平衡区,获得目标机代码空间地址;LTC. W 以源机器代码空间地址为索引、目标机代码空间地址为数据,通过负荷转化通道写负荷平衡区.

当负荷平衡区写缺失时,只需为当前写操作分配新的负荷平衡区缓存空间即可;但是当读缺失发生时,由于无法建立负荷平衡区到处理器内存的映射关系,引入负荷平衡区读缺失寄存器处理读缺失,读缺失发生时,通道读指令返回该寄存器的值. 该方式将负荷平衡区从处理器存储系统独立出来,消除了总线竞争对负荷平衡区读缺失的影响,保证负荷转换通道读写指令可在固定时钟周期内流水完成,有效简化了硬件设计并提高了指令性能.

定义 2. 负荷平衡区读缺失寄存器(read miss register, RMR). 该寄存器的容量为 32 b,负荷平衡区读缺失时返回该寄存器的值.

定义 3. 负荷平衡区读缺失寄存器读写指令(RMR. R/W). RMR. R 读负荷平衡区读缺失寄存器;RMR. W 设置负荷平衡区读缺失寄存器.

2.2 软件算法分析

软件方面利用负荷转换通道实现软硬件协同工作,包括动态翻译器初始化时激活该通道、运行时利用该通道把 ICache 负荷压缩并转移到 DCache 以及退出时关闭该通道.

如何高效利用通道读写指令实现软硬件协同工作是本方法的关键. 本方法需在翻译器初始化时设置负荷平衡区读缺失寄存器的值为负荷转换通道读缺失服务程序(如图 9 所示)入口地址;映射器(如图 10 所示)翻译源程序时把间接转移指令映射为通道读指令和目标机间接转移指令 2 条指令,前者获得目标机代码空间转移目标地址,后者跳转到该地址执行. 该方法有效减少了 QEMU 原始方法处理间接转移指令时执行的目标机指令总数. 据此得到动态翻译器总工作流程,如图 11 所示.

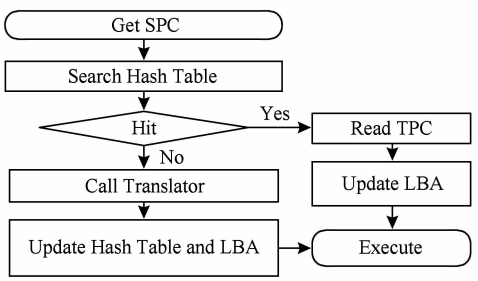


Fig. 9 Read miss handler of LTC.

图 9 负荷转换通道读缺失服务程序

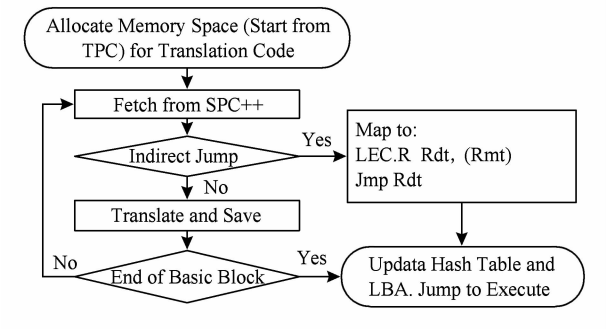


Fig. 10 Work flow of translator.

图 10 映射器流程图

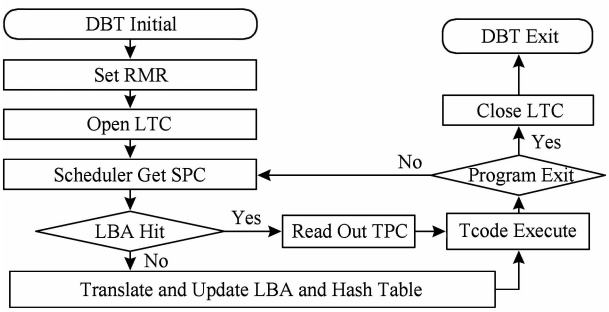


Fig. 11 Work flow for cache load balancing oriented DBT.

图 11 动态翻译器运行流程

通过本算法,调度器调度操作对 ICache 的负荷首先被压缩为负荷平衡区读操作;然后通过内嵌在间接转移指令翻译码中的负荷平衡区读指令将该读操作转移到 DCache,避免间接转移指令执行时执行器和调度器的频繁切换以及调度器地址转换操作对 ICache 的访问负荷,有效提高 DCache 利用率和动态翻译器整体性能.

3 实验与分析

本实验以 ARM Cortex-A 指令集为源指令架构,CSKY 自主指令架构为目标指令架构. 首先通过 Cortex A9 的编译器生成待执行的 EEMBC 测试基准;然后修改 CK810 的软核代码并复用 32 KB DCache 中的 2 KB 作为负荷平衡区实现本方案;最后在 Virtex V5 的 FPGA 上进行本实验. CK810[23] 为杭州中天微系统公司设计的具有自主知识产权的

国产高端嵌入式处理器,其 ICache 和 DCache 各为 32 KB,均采用 4 路组相连结构.

3.1 确定负荷平衡区容量

负荷平衡区主要针对间接转移指令,图 12 分析间接转移指令转移目标地址在不同区间的分布以及不同区间转移总数占总转移次数的百分比.观察发现程序体现出很强的局部性:转移次数大于 15 的转移目标地址占总转移目标地址的 20%,但是这部分转移目标地址的转移总次数却占程序总转移次数的 98%以上.进一步实验发现程序间接转移的目标地址总数不足 300,因此采用 2 KB 的负荷平衡区.

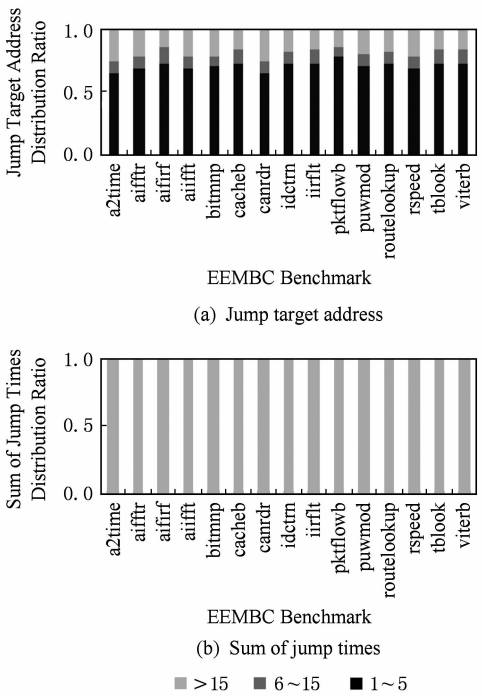


Fig. 12 Distribution of jump target address for indirect jumps in different jump times region.

图 12 间接转移目标地址分布

3.2 缓存负荷分析

图 13 分析本方法调度器、映射器和执行器高速缓存访问负荷的相对比例.ICache 访问负荷方面:调度器 ICache 访问次数几乎被完全优化,只有 pktflowb 程序的调度器存在不足 20%的 ICache 访问次数;进一步对比图 5 和图 13,发现本方法使得 pktflowb 调度器 ICache 访问次数占动态翻译器 ICache 总访问次数的百分比从 39%减少到 14%,ICache 访问次数从 2 470 万次减少到 530 万次.DCache 访问负荷方面:虽然复用部分 DCache 作为负荷平衡区,本方法不但没有为 DCache 性能带来负面影响,反而有效减少了调度器 DCache 访问负荷;几乎所有程序的调度器对 DCache 的访问负荷被完全优化,只有 pktflowb 调度器还存在 25%左右的 DCache 访问负荷;对比图 5 和图 13 发现,本方法使得 pktflowb 调度器 DCache 访问次数占动态翻译器 DCache 总访问次数的百分比从 45%减少到 26%,DCache 访问次数从 2 270 万次减少到 550 万次.

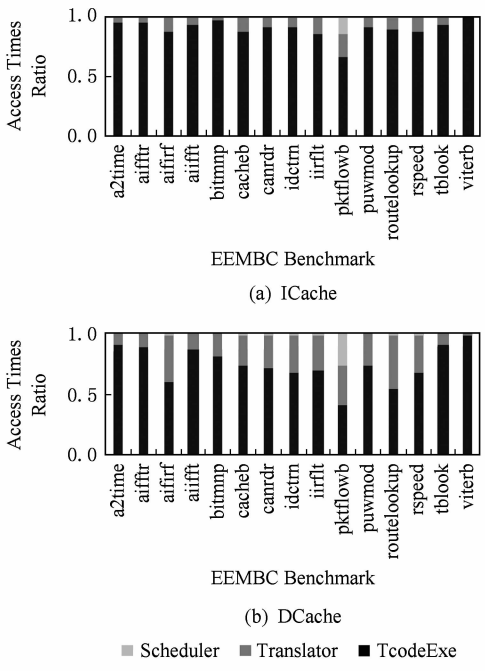


Fig. 13 Distribution of cache access times.

图 13 高速缓存访问次数分布

以处理器配置了 DCache 相比没有配置 DCache 时程序性能提高的百分比作为 DCache 对程序性能的加速比,图 14 以该加速比作为 DCache 利用率,统计了本方法对动态翻译器 DCache 利用率提升的百分比.观察发现 DCache 利用率最小提高 20%,最高提高 650%,平均提高 220%.

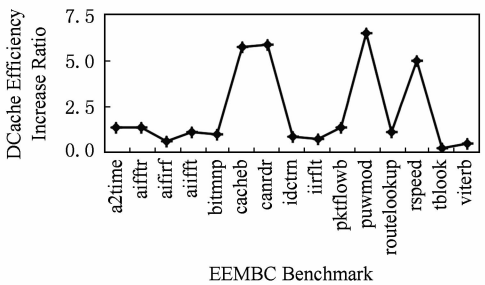


Fig. 14 DCache efficiency increase ratio.

图 14 DCache 利用率提高倍数

图 15 统计了本方法对动态翻译器整体性能提高的百分比,观察发现动态翻译器性能最小提高 12%,最多提高 500%,平均提高 171%.动态翻译器性能提高的百分比类似于 DCache 利用效率提高的

百分比,这是因为本方法对动态翻译器性能的提升主要得益于 DCache 利用率的提升.

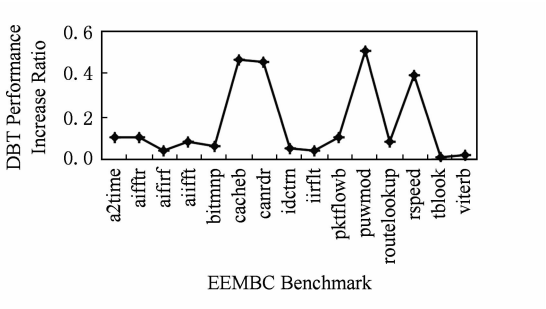


Fig. 15 DBT performance increase ratio.

图 15 DBT 性能提高倍数

3.3 负荷平衡区分析

1) 命中率分析

修改 CK810 软核代码,为其加入负荷平衡区命中率的统计模块,实验发现所有 EEMBC 测试程序负荷平衡区命中率均在 98.5%以上.进一步实验发现,运行 EEMBC 测试程序时,负荷平衡区的缺失率主要来自首次访问导致的强制缺失.

2) 面积开销分析

在 TSMC65LP 工艺下进行综合,发现引入本方法后,CK810 的面积从 2.200 0 mm^2 增加到 2.209 8 mm^2,面积仅增大 0.44%.

4 结 论

本文在分析动态翻译时指令和数据高速缓存访问特性的基础上,提出动态均衡指令高速缓存与数据高速缓存负荷,以平衡高速缓存利用率的软硬件协同翻译方法.该方法通过为处理器设计高速缓存负荷平衡状态和负荷转化通道,将动态翻译器、调度器把源机器代码空间地址向目标机代码空间地址转换时对 ICache 的负荷压缩并转化给 DCache,以提高 DCache 利用率和动态翻译器性能.EEMBC 基准测试实验表明,本方法有效平衡高速缓存利用率,并提高动态翻译器性能.作者接下来将继续研究动态翻译时处理器的高速缓存特性,以进一步提高动态翻译器对高速缓存的利用,来加速动态翻译过程.

参 考 文 献

[1] Li Jianhui, Ma Xiangning, Zhu Chuanqi. Dynamic binary translation and optimization [J]. Journal of Computer Research and Development, 2007, 44(1): 161-168 (in Chinese)
(李剑慧, 马湘宁, 朱传琪. 动态二进制翻译与优化技术研究[J]. 计算机研究与发展, 2007, 44(1): 161-168)

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

[3] Cai Zhanju, Liang Alei, Qi Zhengwei, et al. Performance comparison of register allocation algorithms in dynamic binary translation [C] //Proc of Int Conf on Knowledge and Systems Engineering (KSE'09). Piscataway, NJ: IEEE, 2009: 113-119

[4] Liao Yin, Sun Guangzhong, Jiang Haitao, et al. All registers direct mapping method in dynamic binary translation [J]. Computer Application and Software, 2011, 28(11): 21-24 (in Chinese)
(廖银, 孙广中, 姜海涛, 等. 动态二进制翻译中全寄存器直接映射方法[J]. 计算机应用与软件, 2011, 28(11): 21-24)

[5] Hsu Chunchen, Liu Pangfeng, Wang Chien-min, et al. LnQ: Building high performance dynamic binary translators with existing compiler backends [C] //Proc of Int Conf on Parallel Processing. Piscataway, NJ: IEEE, 2011: 226-234

[6] 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: 1-10

[7] Guan Haibing, Yang Hongbo, Qi Zhengwei, et al. The optimizations in dynamic binary translation [C] //Proc of the 5th Int Conf on Ubiquitous Information Technologies and Applications. Piscataway, NJ: IEEE, 2010: 1-6

[8] Bellard F. QEMU, a fast and portable dynamic translator [C] //Proc of USENIX Annual Technical Conf (USENIX ATC'05). Berkeley, CA: USENIX Association, 2005: 41-46

[9] Probst M, Krall A, Scholz B. Register liveness analysis for optimizing dynamic binary translation [C] //Proc of the 9th IEEE Working Conf on Reverse Engineering. Piscataway, NJ: IEEE, 2002: 35-44

[10] Hong Dingyong, Hsu Chunchen, Yew Penchung, et al. HQEMU: A multi-threaded and retargetable dynamic binary translator on multicores [C] //Proc of the 10th Int Symp on Code Generation and Optimization. New York: ACM, 2012: 104-113

[11] He Yunchao, Chen Kai, Gu Jinghui, et al. A new approach to reorganize code layout of software cache in dynamic binary translator [C] //Proc of the 3rd Int Symp on PAAP. Piscataway, NJ: IEEE, 2010: 175-182

[12] Gu Jinghui, Xu Chao, Lin Ling, et al. The implementation of static-integrated optimization framework for dynamic binary translation [C] //Proc of 2009 Int Conf on Information Technology and Computer Science (ITCS'09). Piscataway, NJ: IEEE, 2009: 510-513

[13] Luk C, Cohn R, Muth R, et al. Pin: Building customized program analysis tools with dynamic instrumentation [C] // Proc of the 2005 ACM SIGPLAN Conf on Programming Language Design and Implementation (PLDI'05). New York: ACM, 2005: 190-200

[14] Ebcioglu K, Altman E, Gschwind M, et al. Dynamic binary translation and optimization [J]. IEEE Trans on Computers, 2001, 50(6): 529-548

[15] Jia Ning, Yang Chun, Wang Jing, et al. SPIRE: Improving dynamic binary translation through SPC-indexed indirect branch redirecting [C] //Proc of the 9th ACM SIGPLAN/SIGOPS Int Conf on Virtual Execution Environments. New York: ACM, 2013: 1-12

[16] Hu Weiwu, Liu Qi, Wang Jian, et al. Efficient binary translation system with low hardware cost [C] //Proc of 2009 IEEE Int Conf on Computer Design (ICCD'09). Piscataway, NJ: IEEE, 2009: 305-312

[17] Hu Weiwu, Wang Jian, Gao Xiang, et al. Godson-3: A scalable multicore RISC processor with X86 emulation [J]. Micro, 2009, 29(2): 17-29

[18] Baiocchi J A, Childers B R, Davidson J W, et al. Enabling dynamic binary translation in embedded systems with scratchpad memory [J]. ACM Trans on Embedded Computing Systems, 2012, 11(4): 1-33

[19] Guha A, Hazelwood K, Soffa M L. Memory optimization of dynamic binary translators for embedded systems [J]. ACM Trans on Architecture and Code Optimization, 2012, 9(3): 1-29

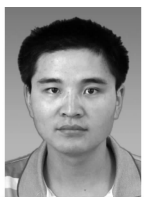
[20] Yue Feng, Pang Jianmin, Han Xiaosu, et al. An improved code cache management scheme from i386 to alpha in dynamic binary translation [C] //Proc of the 2nd Int Conf on Computer Modeling and Simulation. Piscataway, NJ: IEEE, 2010: 321-324

[21] Guha A, Hazelwood K, Soffa M. Balancing memory and performance through selective flushing of software code caches [C] //Proc of the 2010 Int Conf on Compilers, Architectures and Synthesis for Embedded Systems. New York: ACM, 2010: 1-10

[22] He Hongqi, Chen Haifeng, Jiang Liehui, et al. Hardware/software co-design of dynamic binary translation in X86 emulation [C] //Proc of the 2012 Int Conf on Computer Science and Automation Engineering. Piscataway, NJ: IEEE, 2012: 283-287

[23] Hangzhou C-SKY Microsystem Corporation. CK810 Introduction [EB/OL]. (2012-01-01) [2014-01-20]. http://www.c-sky.com/product.php

[24] Hiser J D, Williams D, Hu W, et al. Evaluating indirect branch handling mechanisms in software dynamic translation systems [J]. ACM Trans on Architecture and Code Optimization, 2011, 8(2): 1-28



Li Zhanhui, born in 1988. PhD candidate. His main research interests include dynamic binary translation and high performance embedded micro-computer designing.

Liu Chang, born in 1987. PhD candidate. His main research interests include dynamic binary translation and high performance embedded micro-computer designing(liuchang@vlsi.zju.edu.cn).

Meng Jianyi, born in 1982. PhD, professor. His main research interests include computer architecture studying and high performance embedded micro-computer designing.

Yan Xiaolang, born in 1947. PhD, professor, PhD supervisor. His main research interests include ICCAD flow, computer architecture studying and high performance embedded micro-computer designing(yan@vlsi.zju.edu.cn).