高技术通讯 2012 年 第22 卷 第6 期:583 ~ 588

doi:10.3772/j.issn.1002-0470.2012.06.005

二进制翻译控制转移的软硬件协同设计①

郝守青②[*,**,***,****]　刘　奇③[*,**,***,****]　沈海华[*,**,****]　章隆兵[*,**,****]

([*]中国科学院计算机系统结构重点实验室　北京 100190)

([**]中国科学院计算技术研究所　北京 100190)

([***]中国科学院研究生院　北京 100049)

([****]北京龙芯中科技术服务中心有限公司　北京 100190)

摘　要　针对控制转移开销是影响二进制翻译和优化系统性能的主要因素,进行了提高二进制翻译优化系统性能的研究,提出并实现了硬件设计开销较小的基于硬件内容可寻址存储器(CAM)机制的软硬件协同设计方法。通过实验充分分析了 CAM 大小、软件替换算法对 CAM 命中率的影响,并根据分析提出了一种新颖的、软硬件结合的降低 CAM 访问缺失率的方法。该方法相对于传统的软件和硬件优化方法,硬件实现及验证复杂度低且优化效果明显。实验结果表明该方法使得二进制翻译系统整体性能提高了13.44%。该方法已实际应用于龙芯 x86 二进制翻译系统中。

关键词　软硬件协同设计,二进制翻译和优化,控制转移,龙芯,指令集架构(ISA)

0　引　言

指令集架构(instruction set architecture, ISA)是计算机软硬件的接口。很多现代处理器采用了简洁、高效的精简指令集计算(reduced instruction set computing, RISC)架构。RISC 相对于以 x86 为代表的复杂指令集计算(complex instruction set computing, CISC),结构简单且实现效率高。然而,目前桌面和服务器等领域的主流指令集架构主要采用 x86 及其衍生架构,导致大量优秀的商用软件无法在其它架构的处理器上运行,进而限制了 RISC 处理器在桌面、服务器等领域的推广。二进制翻译和优化系统是实现二进制程序在不同指令集架构处理器上兼容运行的主要途径。

在二进制翻译系统[1,2]中,翻译器以代码块(可以是基本块、超级块等)[3]为单位,将需要执行的代码转换为本地的代码保存下来。如果控制流在同一代码块内则可以得到较好的性能[4];如果控制流进入其它代码块,则需要通过源地址(source program counter, SPC)在映射表中查找对应的目标地址(target program counter, TPC),再根据 TPC 在代码缓存中查找翻译后的二进制代码。以 x86 到 MIPS 的翻译系统为例,软件实现上述查找过程往往需要执行 10 ~ 15 条开销较大的 MIPS 访存和转移指令。因此,针对控制转移开销,前人提出了大量的优化方法。代码块链接技术[5]能够很好地优化直接控制转移。该方法根据翻译时可以得到的跳转目标地址,使用后续地址补丁的方法把代码块链接在一起。大部分二进制翻译和优化系统实现了代码块链接技术。间接跳转预测[6,7]、影子返回地址栈[8,9]等软件方法用来优化间接控制转移。上述方法假定间接跳转的目标地址很少发生改变,通过剖析代码块的执行来预测跳转目标地址。上述方法如果预测失败需要重新查找映射表,优化并不充分。因此,有学者专门设计了跳转翻译快速重编址缓冲器(jump translation lookaside buffers, JTLB)[10,11]和双返回地址栈(double return address stack, DRAS)[12]等硬件装置来加快地址映射表查找。JTLB 每一项包含 tag 标记和 TPC 值,通过哈希 SPC 来访问 JTLB。为了进一步提高性能,JTLB 还可以跟分支目标缓冲器(branch target buffer, BTB)结合起来。处理器在分

① 863 计划(2008AA010901),国家自然科学基金(60736012,60921002,61070025)和 973 计划(2005CB321600)资助项目。

② 女,1982 年生,博士生;研究方向:多核处理器结构,计算机系统性能和验证;E-mail: haoshouqing@ict.ac.cn

③ 通讯作者, E-mail: liuqi@ict.ac.cn

(收稿日期:2010-12-17)

支预测阶段就开始执行 JTLB 的查找，减少流水线停顿。DRAS 技术可与超标量处理器中的返回地址栈(return address stack, RAS)[13]结合起来。结合后当函数返回指令被取指令时，DRAS 预测后续地址的 SPC 和对应的 TPC；当函数返回指令被执行时，比较预测的 SPC 值与实际返回的 PC 值是否相同。上述硬件方法能够加速地址查询，但因其需要遍历流水线，硬件实现及验证的复杂度高，且当 JTLB 或者 DRAS 不命中时，需要执行相应的不命中函数，会带来相当大的开销。针对上述软件预测方法的不准确性及硬件方法实现的复杂性，本文研究并实现了硬件设计开销较小的软硬件协同优化方法来降低间接控制转移开销。该方法的正确性和性能在基于龙芯的 x86 二进制翻译平台[14,15]上得以验证。

1 控制转移的主要类型和比例

控制转移主要是由跳转指令引起的。以 x86 指令集[16]为例，跳转指令主要包括 CALL，Jcc，JMP，LOOP/LOOPcc 和 RET 等。它们的主要功能和操作数如下：

(1)CALL 指令实现过程调用，主要完成在栈上保存过程链接信息，根据操作数跳转到目的地址。目的地址操作数可以是立即数、通用寄存器或者内存地址。

(2)Jcc 指令实现条件检查并完成跳转功能，主要完成检查 EFLAGS 寄存器中一个或者多个状态的值，如果符合一定的条件则跳转到操作数指示的目的地址。目的地址操作数主要是一个 8 位、16 位或 32 位的偏移地址。

(3)JMP 指令实现了不保存返回地址信息的跳转功能。目的地址操作数可以是立即数、通用寄存器或者内存地址。

(4)LOOP/LOOPcc 指令把 ECX/CX 的值作为计数器实现循环。每次 LOOP 指令执行后，将计数器寄存器的值减 1，并检查是否等于零。如果计数值为零，循环终止，继续执行下面的指令；如果计数值不为零，则跳转到循环开始的位置。跳转目的地址是一个 8 位的相对偏移。

(5)RET 指令实现把控制交回栈顶返回地址的功能。RET 指令有一个可选参数，可以用来实现释放栈顶开始的若干字节的功能。

根据目的地址操作数的不同，上述跳转指令可以分为直接跳转指令和间接跳转指令。Jcc、LOOP/LOOPcc 属于直接跳转指令，其目的地址通过地址偏移等方式直接给出；RET 属于间接跳转指令，其目的地址保存在寄存器或者内存中；CALL 和 JMP 根据目的地址操作数的不同有可能是直接跳转指令也有可能是间接跳转指令。根据剖析结果可知在 x86 处理器执行 SPEC CPU2006[17]过程中，直接跳转和间接跳转的比例分别为 87.79% 和 12.21%，如图 1 所示。

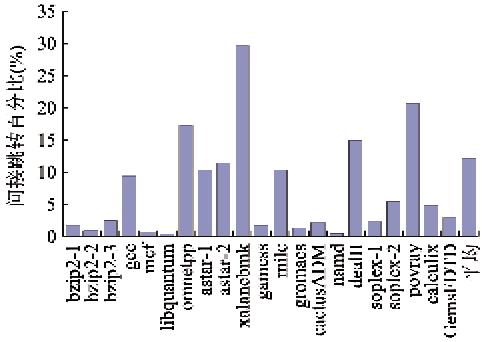

图 1 间接跳转指令占总跳转指令的比例

从上述统计信息分析得知：直接跳转的比例大，需要进行有针对的优化；间接跳转的比例相对较小，然而每次处理开销巨大，因此也需要进行优化。对于直接跳转类指令，其目的地址在被翻译时可知，因此代码块链接优化技术能够很好地降低其开销。而间接跳转类指令，其目的地址只有在被翻译后且执行时才可知，因此成为影响二进制翻译和优化系统性能的主要因素。

以 x86 指令集到龙芯平台的翻译为例，本文在龙芯 x86 二进制翻译平台上实现了代码块链接技术[5]。实验结果如图 2 所示，代码块链接方法能够有效优化直接控制转移开销，使整个系统性能平均提高了约24.64%。

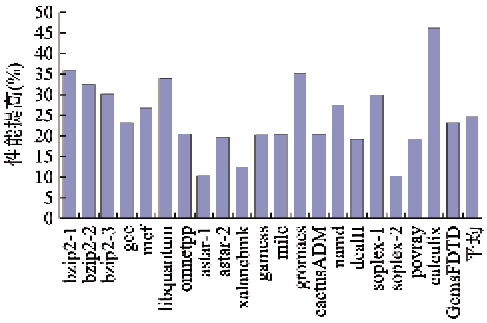

图 2 代码块链接优化的性能提高

2 基于 CAM 的软硬件协同优化设计

2.1 实现间接跳转的 CAM 硬件优化

以 x86 指令集到龙芯平台的翻译为例,对于 JMP/CALL、RET 等间接跳转指令,二进制翻译系统在翻译的过程中无法预测保存在寄存器或者内存中的 SPC 值,因此不能使用代码块链接技术。一般而言,当执行过程中遇到间接跳转指令时,需要实时查找地址映射表得到 SPC 对应的 TPC 的值,从而把控制转移到下一个代码块的位置。因此,加速 SPC 到 TPC 的映射过程是高效翻译和执行间接跳转指令的关键。

为优化间接跳转指令,本文在处理器中增加了如图 3 所示的内容可寻址存储器(content-addressable memory, CAM)部件[18],用以加速代码翻译过程中间接跳转地址 SPC 到 TPC 的转换。该硬件 CAM 每项拥有 8 位 asid、40 位 CAM _ value 和 64 位的 RAM _ value。处理器提供了如表 1 所示的 3 条指令用于 CAM 的访问。二进制翻译管理代码访问该 CAM 表时,硬件 CAM 并行比较各个表项中的 CAM _ value 值,如果命中仅仅需要 2 条指令开销就可以得到映射后 TPC 的值,不需要访问内存,极大地提高了效率;如果缺失,则需要重填 CAM,重填等开销会导致一定的性能损失。因此 CAM 表的命中率是影响 CAM 性能的重要因素。

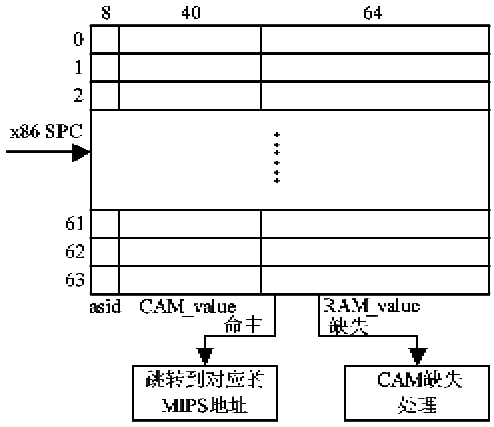

图 3 硬件 CAM 机制

表 1 用于访问 CAM 的指令

指令	作用
CAMPI	根据索引查找 CAM 表
CAMPV	根据值查找 CAM 表
CAMWI	根据索引写 CAM 表

为降低 CAM 缺失率,一种直观的方法是增加 CAM 项数。经评估发现,如图 4 所示,64 项 CAM 表相对于 32 项,能够明显降低缺失率,但是 128 项相对于 64 项效果不明显。超过 64 项,除非大规模增加 CAM 项数,否则很难有效提高 CAM 命中率,但是大规模增加 CAM 项数带来的物理设计复杂度及硬件开销使得该方法不切实际。基于上述评估结果,本文实现了使用 64 项的 CAM。

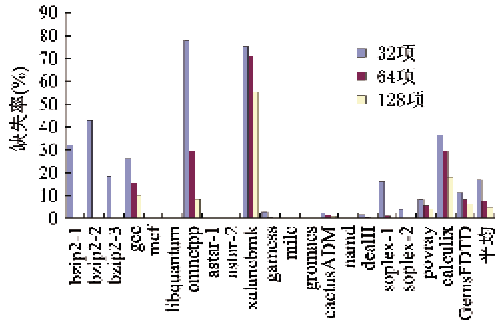

图 4 CAM 项数与缺失率的关系

2.2 CAM 替换算法分析

CAM 表的设计一旦确定下来,影响命中率的首要因素是 CAM 项的替换算法[3]。CAM 替换算法主要有最近最少使用替换算法,满则清空算法,预测式清空算法,先进先出算法,随机替换、轮转替换算法等。根据算法实现的复杂度和 CAM 结构的特点,本文实现了满则清空、先进先出和轮转随机这 3 种替换算法,并对其进行了性能评估,结果如图 5 所示。从实验结果可以看出,除了满则清空算法的缺失率较高外,其它算法的命中率均差别不大。这是因为二进制翻译和优化执行的代码块有很强的局部性,如果采用一次性清空算法,很可能会把大量有用的映射清除出 CAM 表;而其它算法对程序执行的路径都没有准确的预测,导致的 CAM 缺失率不会有很大区别。考虑到算法实现的复杂度,本文实现时采用了基于轮转的随机替换算法。

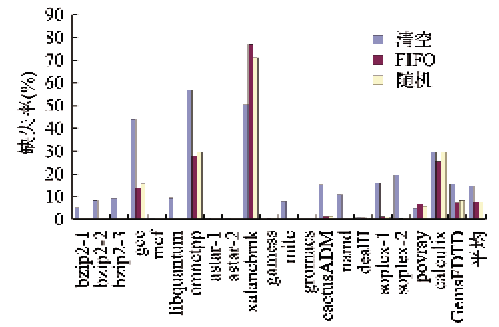

图 5 不同 CAM 替换算法导致的缺失率比较

2.3 CAM 缺失原因分析及软硬件协同优化

为了进一步提高间接跳转的翻译和执行效率，本文对引起 CAM 缺失的主要原因进行了分析。x86 间接跳转指令主要包括 JMP/CALL 指令和 RET 指令。我们统计了这两种间接跳转指令导致的 CAM 缺失的比例，结果如图 6 所示，统计结果显示 85.80% 的 CAM 缺失是由 JMP/CALL 类指令导致的。通过进一步分析原因，我们发现 JMP/CALL 指令其绝对数量要比 RET 类指令多很多。这是因为：首先，在一个函数内部因控制流的跳转存在大量 JMP 指令是正常的；其次，对于现代编译器而言，函数内联优化进一步减少了函数调用的数量，相应地减少了 RET 指令的数目。在 SPEC CPU2006 测试程序中，函数调用（CALL/RET）静态统计的数目仅为 JMP 类指令的 5% 左右。综上可知，CAM 缺失主要是由 JMP 类指令导致的。

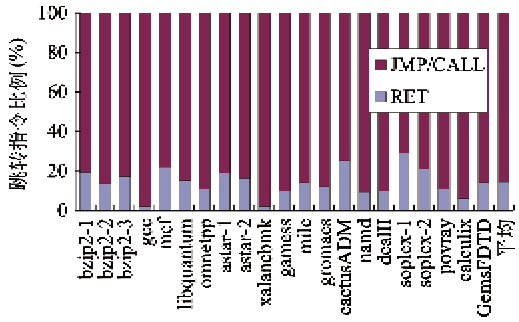

图 6 不同间接跳转指令类型导致 CAM 缺失的比例

为了进一步提高 CAM 表的命中率和降低开销，本文提出了一种软硬件协同使用 CAM 表的算法。该算法的思想是分离不同种类的间接跳转指令，针对不同的跳转指令采用不同的软硬件优化算法，降低 CAM 表项发生冲突的几率，从而提高系统的整体性能。该算法流程如图 7 所示。

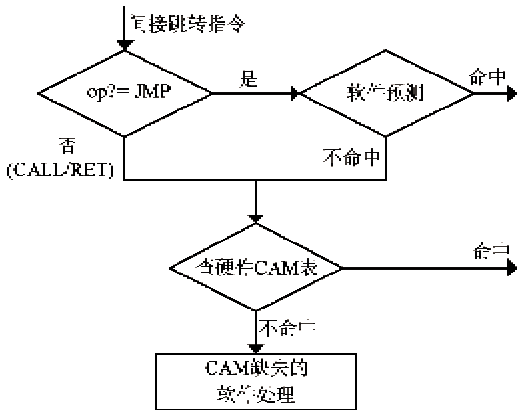

图 7 软硬件协同优化算法流程图

该算法使用软件预测技术来优化 JMP 类型指令导致的间接跳转，对其它情况导致的间接跳转，如 JMP 指令预测失误、CALL/RET 指令等，直接查找硬件 CAM 表。如果 CAM 表访问缺失，则访问由软件维护的 SPC 到 TPC 的总映射表。软件预测 JMP 类指令的方法如下：首先，在代码块的结束位置填入若干空预测槽位，用来填入相应的预测值。当程序执行到代码块的结束位置时，如果检测到还有多余的预测槽位就把查表得到的 SPC 和 TPC 填入预测槽位中；如果下次需要跳转到以前跳转过的 SPC 地址，可以从预测槽位中直接得到对应的 TPC 值，不需要查找 CAM 表。

该方法不需要通过大量剖析来预测最优的 SPC 和 TPC 预测项，因为如果一旦预测错，还可以通过快速的 CAM 表查询得到对应的 TPC。本文评估了不同预测槽位数目导致的 CAM 缺失率的变化情况，如图 8 所示，实验结果表明当预测槽位数目为 2 时就能够维持较低的 CAM 缺失率。如果预测槽位数目过多，会加大比较开销，反而会降低系统整体性能。

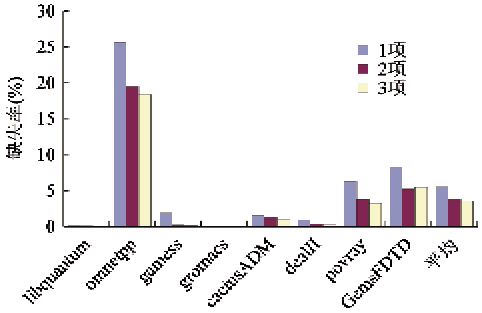

图 8 不同软件预测槽位数目导致 CAM 缺失的比例

3 实验结果和分析

基于龙芯 x86 二进制翻译和优化平台[14,15]，本文实现了基于 CAM 的软硬件协同优化算法，并对其进行了性能评估。统计结果显示该方法使用两项预测槽，将 CAM 的缺失率平均降低了 56.32%。实验结果如图 8 所示。在 SPEC CPU2006 中，soplex、calculix 等原先 CAM 缺失率非常高的程序缺失率降低非常明显，分析后发现这些程序的核心循环存在大量的函数内跳转，这些跳转主要是由 JMP 指令实现的，而本文使用的软硬件协同优化算法有效减小了此类跳转指令给 CAM 表带来的压力。除此之外，xalancbmk 等 CAM 缺失率较高的程序缺失率下降幅度并不明显，这主要是因为这些程序存在大量的

CALL/RET 指令,只有 JMP 的软件优化带来了少许的 CAM 缺失率降低。

本文还统计了软硬件协同优化算法对系统整体性能的影响,如图 9 所示,该方法使得系统整体性能平均提高了 13.44%。从统计结果可知,CAM 缺失率的下降为大部分程序带来了性能的提高:soplex、calculix 等 CAM 缺失率下降明显的程序,其性能均有较大幅度提高;xalancbmk 程序的 CAM 缺失率虽略有下降,但是整体性能并没有任何提高,主要是因为软件算法替代硬件 CAM 表查找带来的开销导致的;mcf 程序虽然优化前的 CAM 缺失率已经非常低了,但是优化算法仍然降低了 CAM 缺失率,并带来了非常大的性能提高。二进制翻译器的剖析结果证明 mcf 本身核心循环很小,各方面开销也几乎不存在,间接跳转导致的 CAM 表缺失是该程序性能低下的主要原因。

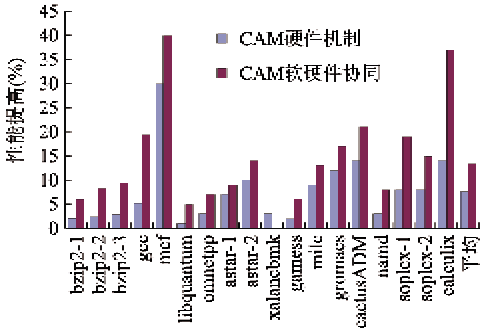

图 9　间接跳转优化算法带来的性能提高

此外,图 9 还比较了本文提出的软硬件协同优化算法和传统单纯硬件 CAM 算法的性能。从统计结果可以得知,相对于传统纯硬件 CAM 机制,本文的协同优化算法能够带来更大的性能提高。其中 bzip2、soplex、calculix 等程序具有相当大的性能提高。这是因为跟 CAM 机制配合的软件算法能够有效降低 CAM 缺失率;而 mcf 性能提高幅度有限是因为 mcf 本身 CAM 缺失率就比较低。

本文提出的硬件 CAM 机制相对于 JTLB、DRAS 等硬件方法,结构设计简单、验证复杂度低。经过初步的物理设计评估,其硬件面积开销仅为 JTLB 方法的 40% 左右,同时 CAM 机制没有复杂的硬件逻辑支持,可以进一步大幅度降低硬件设计和验证开销。效率方面,由于访问 CAM 时不需要遍历流水线,不会像 JTLB 或 DRAS 方法那样造成流水线堵塞、停顿,访问效率较高。并且,该机制配合相应的软件预测方法,仅使用两项预测槽就可以将访问缺失率降低 56.32%。即使软件预测失败,访问硬件 CAM 的延迟也很小。目前,本文提出的基于 CAM 的软硬件协同优化方法已在龙芯处理器 x86 二进制翻译和优化平台中得到了实际应用。

4　结　论

控制转移是影响二进制翻译和优化系统性能的主要因素。为了降低控制转移开销,提高二进制翻译优化系统的性能,本文提出并实现了基于 CAM 的软硬件协同优化方法。本文通过具体实验详细地分析了 CAM 大小、替换算法、缺失率等因素对系统整体性能的影响,并提出相应的软件预测方法以提高硬件 CAM 机制的效率。依托龙芯 x86 二进制翻译和优化系统平台,我们实现并验证了基于 CAM 的软硬件协同优化方法的有效性。实验数据表明,本文提出的基于 CAM 的软硬件协同优化方法能够有效降低间接控制转移开销,提高系统整体性能。

参考文献

[1] Altman E R, Kaeli D, Sheffer Y. Welcome to the opportunities of binary translation. *IEEE Computer*, 2000, 33: 40-45

[2] Altman E R, Ebcioglu K, Gschwind M, et al. Advances and future challenges in binary translation and optimization. *Special Issue on Microprocessor Architecture and Compiler Technology, Proceedings of the IEEE*, 2001, 89(11):1710-1722

[3] Smith J E, Nair R. Virtual Machines: Versatile Platforms for Systems and Processes. San Francisco: Morgan Kaufmann Publishers, 2005

[4] Drongowski P J, Hunter D, Fayyazi M, et al. Studying the performance of the FX! 32 binary translation system. In: Proceedings of the 1st Workshop on Binary Translation, Newport Beach, USA, 1999

[5] Piumarta I, Riccardi F. Optimizing direct threaded code by selective inlining. In: Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, New York, USA, 1998. 291-300

[6] Deutsch L P, Schiffman A M. Efficient implementation of the Smalltalk-80 system. In: Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, New York, USA, 1984. 297-302

[7] Bala S B V, Duesterwald E. Transparent Dynamic Optimization: The Design and Implementation of Dynamo: [Technical Report]. Cambridge: HP Laboratories, 1999

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

[9] Gschwind M K. Method and Apparatus for Rapid Return Address Computation in Binary Translation: [Technical Report]. IBM Research Disclosures YOR819980410, 1998

[10] Kim H S, Smith J E. Hardware support for control transfers in code caches. In: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, Washington DC, USA, 2003. 253

[11] Fahs B, Bose S, Crum M, et al. Performance characterization of a hardware mechanism for dynamic optimization. In: Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture, Washington DC, USA, 2001. 16-27

[12] Kaeli D R, Emma P G. Branch history table prediction of moving target branches due to subroutine returns. In: Proceedings of International Symposium on Computer Architecture, Washington DC, USA, 1991. 34-42

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

[14] Hu W W, Liu Q, Wang J, et al. Efficient binary translation system with low hardware cost. In: Proceedings of IEEE International Conference on Computer Design, Lake Tahoe, USA, 2009. 305-312

[15] 胡伟武,张福新,李祖松. 龙芯 2 号处理器设计和性能分析. 计算机研究与发展, 2006, 43: 959-966

[16] Intel 64 and IA-32 Architectures Software Developer's Manuals. http://www.intel.com/products/processor/manuals/: Intel, 2010

[17] Henning J L. SPEC CPU2000: measuring CPU performance in the new millennium. *Computer*, 2000, 33(7): 28-35

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

A hardware software co-design method for control transfer optimization in binary translation system

Hao Shouqing[*,**,***,****], Liu Qi[*,**,***,****], Shen Haihua[*,**,****], Zhang Longbing[*,**,****]

([*] Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] Graduate University of Chinese Academy of Sciences, Beijing 100049)

([****] Loongson Technology Corporation Limited, Beijing 100190)

Abstract

In view of the fact that the cost of control transferring is of great influence on the performance of binary translation and optimization systems, the study was conducted and a novel hardware software co-design approach based on the content-addressable memory (CAM) was proposed and implemented to reduce the overhead of control transferring. The CAM based method was designed to optimize the indirect branch. The influences of the CAM size and the software replacement algorithm on the CAM's impact rate were thoroughly analyzed by experiment. Based on that, a novel soft-hard combination method for reducing CAM's access absence rate was proposed. The proposed approach can reduce the design and verification costs. Besides, it achieves good performance. The experimental results show that the system performance can be improved by 13.44% with it. The approach is applied to the Godson-3 binary translation system.

Key words: hardware software co-designed, binary translation and optimization, control transferring, Godson, instruction set architecture(ISA)