doi:10.3772/j.issn.1002-0470.2014.01.004

基于二进制插桩的共享指令集异构多核处理器进程迁移方法①

刘宏伟②[*,**,***,****] 邱 吉[*,**,***,****] 高 翔[*,**,****] 陈云霁[*,**,****]

([*]中国科学院计算机体系结构国家重点实验室 北京 100190)
([**]中国科学院计算技术研究所 北京 100190)
([***]中国科学院大学 北京 100049)
([****]龙芯中科技术有限公司 北京 100190)

摘 要 研究了异构多核处理器进程迁移的特点,针对目前解决共享指令集异构多核处理器异构多核间进程迁移方法存在效率、代价、兼容性或者可编程性上的不足,提出了一种基于二进制插桩的进程迁移方法,该方法能够充分利用共享指令集异构多核的优势,以很低的代价大大提升运行效率,并且无需修改源代码和编译系统,有良好的兼容性。在SPEC 等测试程序上的实验数据表明,这种方法的效率为内核模拟的2.25 倍。

关键词 共享指令集(ISA),异构多核处理器,进程迁移,二进制插桩,内核模拟

0 引 言

异构多核处理器相比于同构多核处理器在性能、能量效率以及面积效率上都更有潜力,因而得到了学术界和工业界的广泛关注。目前已有多款异构多核产品问市,如 ARM 的 Tegra、飞思卡尔的 Vybird[1]、AMD 的 Fusion 芯片[2]、Philips 的用于数字机顶盒的 Viper PNX-8500[3],以及 IBM、东芝和索尼公司联合开发的 CELL[4,5] 等。异构多核处理器有完全不同指令集(instruction set architecture,ISA)异构多核处理器和单一 ISA 异构多核处理器。前者尽管显现出更高的性能,但难以编程。后者由于每个核上的 ISA 完全相同,编程相对简单,但失去了利用某些核上专有的特殊指令进行加速的机会。共享指令集(shared ISA)异构多核处理器则同时提供了可编程性和特殊指令集的优势。共享指令集异构多核处理器使用一个基本的指令集,高性能大核拥有某些额外的特殊指令和更高的性能;低功耗小核有基本指令集和更高的能量效率。为了有效而透明的使用共享指令集异构多核处理器,必须实现高效的异构多核间的进程迁移(execution migration)。

目前解决异构多核间进程迁移的方法分为两类,一类是解释执行,如二进制翻译[6],内核模拟;另一类是本地执行,如仅使用基本指令集、胖二进制[7]、程序员定制多版本的代码、以及出错后迁移[8]。这些方法存在效率、代价、兼容性以及可编程性的问题。针对这些问题,本文提出了基于二进制插桩的进程迁移方法。该方法以较小的代价实现了共享指令集异构多核处理器间的进程迁移,增大了任务的可调度空间,为系统进行更高层次的优化提供了良好支持。

1 异构多核处理器进程迁移方法

异构多核处理器间的进程迁移一般发生在如下情况:负载均衡,进程迁移到相对空闲的核上;高优先级,需要更高运算性能的进程进入运行队列;设备电源状态改变,如电池设备进入低电量状态,进程从大核迁移到小核(大核可能从此被关闭);程序进入不同计算需求的阶段,如当前特殊指令密集的进程迁移到大核上去,特殊指令相对稀疏的进程迁移到小核上去;芯片部分过热,进程迁移到温度较低的核上。

实现进程迁移的解释执行方法在二进制翻译上存在启动时间长、执行效率低、空间代价大等问题,难于实用。而内核模拟则适用于共享指令集异构多

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003,2009ZX01029-001-003, 2010ZX01036-001-002)和国家自然科学基金(60921002)资助项目。
② 男,1984 年生,博士生;研究方向:计算机体系结构,处理器设计,VLSI 设计;联系人,E-mail:liuhongwei@ict.ac.cn
(收稿日期:2013-02-21)

核处理器。其上操作系统使用基本指令集，小核上运行特殊指令时则由内核模拟，这种方法简便有效，系统修改代价也最小，但频繁进出内核以及由此引发流水线清空，会影响运行效率。

进程迁移的本地执行方法是指进程迁移到目标处理器核上后以本地机器码执行。这种方法的复杂性在于其要求应用程序、操作系统以及编译系统多方配合。应用程序需要提供所支持的目标处理器核的多个代码段、初始化数据以及辅助迁移的信息，部分方法还需要直接修改源代码以配合进程迁移[7]；操作系统需要提供进程迁移检查点的判断，完成进程状态的转换，以及必要的解释执行的支持[9]；编译系统则需要支持多目标编译、检查点插桩以及核间迁移专门优化[6,10]。目前这方面已有不少研究工作，但得出的方法都需要对系统做大量的修改，有些还需要提供其它系统组件，如专门的迁入和迁出进程来负责转换进程状态[11]。同时这类方法对迁移时机也做出了限制，大多数要求在检查点才可以迁移，不仅代价较大而且降低了迁移的灵活性。

基于以上分析，并结合共享指令集异构多核处理器的特点，本文提出了基于二进制插桩的共享指令集异构多核处理器进程迁移方法。该方法选择在进程迁移到小核而且遇到特殊指令时对原代码进行二进制插桩，生成插桩后代码上下文存放于单独的内存区域，避免了胖二进制方法对空间的固定占用。插桩操作仅针对特殊指令，对数据段、堆、栈没有影响，状态转换代价小。同时无需修改源代码或者编译系统，也不需要借助其它工具，兼容性好。内核管理上，尽可能地降低发生二进制插桩或者状态转换发生的机会，实现了进程的无缝迁移。实验证明，该方法在不同核间迁移时，二进制插桩的平均时间是 191ms，进程转换的平均时间是 0.73ms，运行效率为内核模拟的 2.25 倍。

2 基于二进制插桩的异构多核进程迁移方法

2.1 系统设计

进程在共享指令集异构多核处理器间迁移时，二进制插桩以及进程状态的转换由操作系统触发并完成。为最大化性能，二进制插桩保留原始代码段，插桩后的代码置于 Codebuffer 中。插桩操作仅执行一次，此后再迁移时，只要在两个代码段间进行地址映射即可。

由于共享指令集异构系统中不同核间存在着结构差异，比如寄存器的种类、数量、处理器状态标志等。因此进程的内存映像不仅要保存进程的公共状态，还要分别保存大核和小核特有的状态。在进程切换时，操作系统根据源和目标处理器核分别保存和恢复的各自特有的进程状态，如图 1 所示。

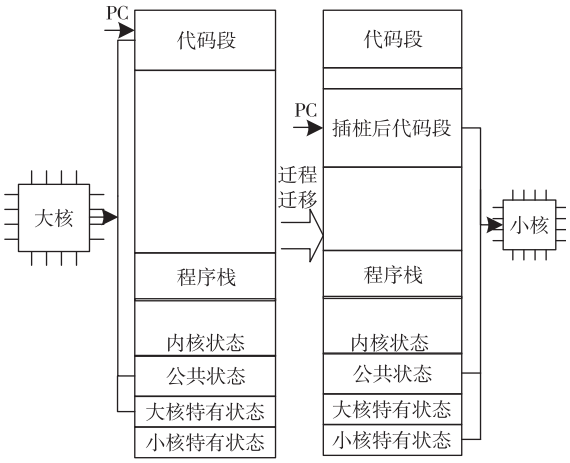


图 1 含特殊指令进程从大核迁移到小核

2.2 插桩后代码运行环境

图 2 给出了运行环境框架。插桩后的代码作为进程在小核上执行的代码段，置于 Codebuffer 中。

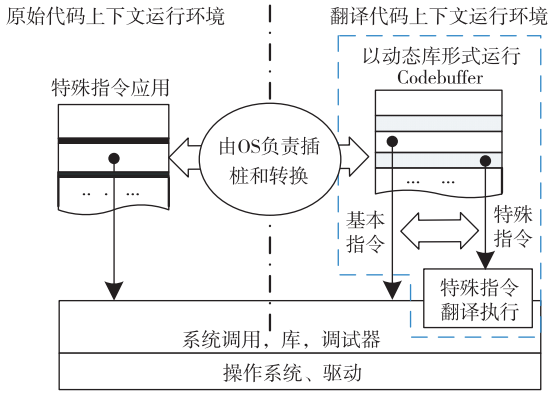


图 2 使用插桩的二进制翻译方法进行进程迁移的基本框架

为了区分，这里把应用程序最初的运行环境叫作原始代码上下文运行环境，把二进制插桩后的代码运行环境叫做插桩代码上下文运行环境。Codebuffer 以及用于支持特殊指令解释执行的函数以动态库的形式运行，这样保证 Codebuffer 私有的同时，用于解释执行的代码段又可以被所有进程共用。插桩代码的生成以及同原始代码上下文间的转换，由内核提

供支持。

2.3 基于二进制插桩的执行方法

本文使用二进制插桩的方法生成翻译代码。插桩时对代码执行一次遍历,根据指令类型进行区分,基本指令集指令直接拷贝到 Codebuffer 中,特殊指令在插入用于转移控制的桩代码后再拷贝。整个插桩过程仅对代码执行一次遍历。若设代码段长度为 N,则二进制插桩的空间复杂度和时间复杂度均为 $O(N)$。这种插桩代价相对于进程总体运行时间来说很短,对整体性能的影响不大,但作为迁移代价,这是开销相对较大的部分,所以在进程迁移时要尽量减少启动二进制插桩的可能。

为降低插桩操作导致的代码膨胀,桩代码要尽可能简洁,所以此处的桩代码仅完成控制转移,执行模式转换、解码及执行则在动态库中完成。插桩代码上下文环境中特殊指令的处理流程见图 3。

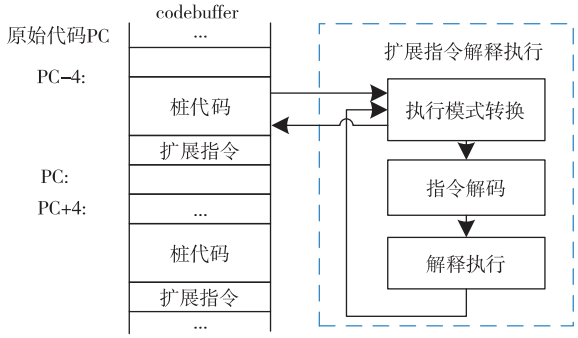


图 3 基础指令和扩展指令之间的切换

2.4 内存映像一致性

进程的内存镜像在不同的核上存在不一致,如变量、函数地址、程序栈、堆等的地址排布不同,它们在处理器核间迁移时,就需要对内存映像做出修改,这就造成迁移开销。因此要使快速进程迁移成为可能,这就需要尽可能保证进程在不同的处理器核上执行时,这些项目都在相同的虚拟地址上。

由于二进制插桩方法以及共享指令集异构多核处理器自身的优势,保证内存镜像一致性显得相对容易。二进制插桩操作对数据段、堆和程序栈没有任何影响,因而原始代码上下文和插桩代码上下文中对数据、堆和栈的虚拟地址的访问都是一致的,满足了数据一致性、堆一致性和栈一致性。实现内存映像一致性,需要考虑的就只有代码一致性了。而代码一致性的保证,则是通过在插桩翻译时,对跳转指令的修改实现的。这样在二进制插桩后的进程迁移,只需要修改程序 PC 值,迁移代价很小。

2.5 迁移过程

操作系统实施共享指令集异构多核处理器间的进程迁移时,需要关注插桩时机的选取和进程状态的切换。根据之前的分析,二进制插桩时间相对进程状态转换时间要长,因此翻译时机的选取对迁移效率影响更大。

为了尽量降低发生二进制插桩的可能,插桩时机选择在不得不插桩的时刻。即在遇到特殊指令引发例外时,再进行插桩并进行进程状态转换。相比于调度时插桩,这种方法避免了进程在小核上运行期间并未遇到特殊指令却执行插桩而产生无谓开销的情况。这种懒惰的插桩时机,可以最大程度保证系统效率。

基于同样的考虑,在获得插桩代码上下文以后,进程状态转换使用相同的方法。图 4 显示了进程迁移时进程在不同核间不同上下文的转换过程。

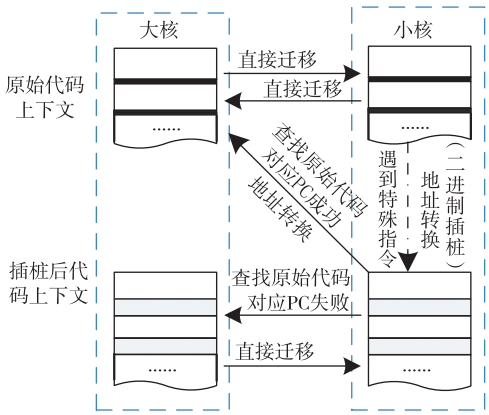
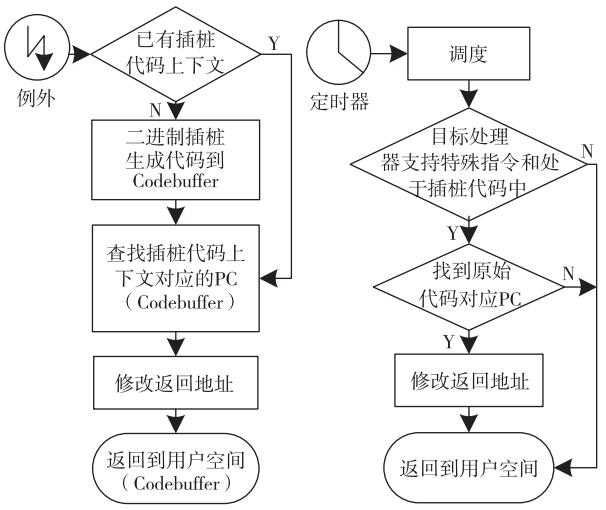


图 4 不同核之间上下文的转换

获得插桩代码上下文之后,进程从大核迁移到小核相对简单,由操作系统将进程的原始代码 PC 值映射成插桩后代码 PC 值并实现转换,流程如图 5 中左图所示。

从小核迁移到大核,需要针对迁移发生时所处上下文的分别处理。若处在原始代码上下文中,说明小核执行期并未遇到特殊指令,则直接进行迁移。若处于插桩代码上下文中,则需要进行 PC 映射。映射时,查询插桩时建立的程序地址表并扫瞄当前函数,根据函数内偏移查找对应 PC,查找成功则返回原始代码上下文;查找失败则留在插桩代码上下文,在以后的迁移中,再寻合适时机转到原始代码上下文。查找失败的发生一般是由于迁移发生时,进程于动态库或者桩代码中。从小核到大核的迁移过

程如图 5 右栏所示。



左：程序从支持特殊指令的大核迁移到基本指令集的小核
右：程序从基本指令集的小核迁移到拥有特殊指令的大核

图 5 共享指令集异构多核处理器进程迁移过程示意图

3 实验平台及性能分析

3.1 实验环境

以龙芯 3B780E 开发板作为实验平台，在此平台上实现了基于二进制插桩的异构多核进程迁移。

实验使用的处理器龙芯 3B 是一款集成 8 个 GS464V 处理器核的多核芯片。其处理器核 GS464V 是由中国科学院计算技术研究所研制的一款 64 位，四发射，超标量处理器核，兼容 MIPS64 指令集，同时具有龙芯自定义的 256 位向量指令扩展，主要面向科学计算，音视频解码等应用，作为多核处理器中的高性能核使用。为了实验方便，低功耗小核通过关闭 GS464V 的向量指令集支持来实现。同时关闭其它处理器核，只保留两个核作为高性能核和低功耗核使用。操作系统运行于 MIPS64 指令集上，向量指令做为大核专有的扩展指令使用。

一个高效的进程迁移方法，要同时提供较高的执行效率、较小的迁移代价以及很好的兼容性和可编程性。对于共享指令集异构多核处理器而言，内核模拟同时具有以上优势，因此本文选择内核模拟作为对比方法，以评估性能。

本研究按表 1 对两种共享指令集异构多核处理器进程迁移方法在机制和功能上进行对比。

表 1 迁移方法对比

迁移方法	二进制插桩	内核模拟
基础指令	本地执行	本地执行
扩展指令	解释执行	解释执行
扩展指令是否需要进入内核态	否	是
任务迁移时是否需要额外内核支持	是	否

3.2 目标应用程序

选取两类测试程序。一类是标准测试集，如 SPEC2000 中若干个测试程序、CBLAS 数学库以及 Stream 微基准测试，这些测试程序依靠支持向量指令的编译器自动生成向量指令。另一类是应用向量指令专门开发的库函数，如 fft、fir 和 iir。

SPEC CPU2000 是 SPEC（标准性能测试协会）开发的专门用于评价 CPU、存储层次和编译性能的一套测试程序。其中的测试程序均从真实的应用中提取，客观而具有代表性，是目前评价处理器和编译器性能最常用的测试程序；Stream 是由弗吉尼亚大学的 John McCalpin 教授开发的测试集程序，主要测试系统的实际访问主存性能。CBLAS 是 BLAS 库的 C 语言接口，它是一种应用程序标准接口，包含大量已编好的关于线性代数运算的程序。主要用于高性能运算领域，本文利用高并行度计算基准测试（High performance Computing Linpack Benchmark，HPL）对 CBLAS 库进行测试。

本研究使用的生成特殊指令的方法基于 GCC 的自动向量化，因此选取向量化比例较高的测试程序来评估性能。表 2 统计了各测试程序中向量化循环数目以及所占比例。

表 2 向量化循环以及比例

测试程序	循环总数	向量化循环数	向量化比例（%）
172. grid	59	5	8. 5
173. applu	171	16	9. 4
178. galgel	640	114	17. 8
252. eon	231	28	12. 1
254. gap	1653	74	4. 5
256. bzip2	159	17	10. 1
CBLAS 核心函数 dgemm	20	8	40

3.3 迁移代价

迁移代价指进行二进制插桩的时间和进程状态

转换时间。针对所选择的测试程序,首先测量了二进制插桩的平均时间,之后在两个方向上各进行了1000 次进程状态转换,得到了平均转换时间。

从表 3 中可以得出,首次代码翻译平均时间为191.18ms,从原始代码上下文到翻译后代码上下文的平均转换时间为 0.73ms,从翻译后代码上下文到原始代码上下文的平均转换时间为 0.72ms。文献[7]实现了在 MIPS 和 ARM 两种核间的进程迁移,其平均转换时间分别为 272ms 和 344ms,其中 MIPS 和 ARM 两种处理器核分别运行在 2GHz 和 833MHz 上。相比之下,本文提出的方法,在评估平台的运行平台的频率较低的情况下,首次代码翻译的时间仍小于其转换时间。而在此后的转换中,本文方法的平均转换时间则更显优势。

一般内核的负载均衡时间间隔约为 200ms,两次时钟中断的时间间隔为 4ms。因此,翻译时间是可以接受的,平均转换时间对系统的影响也很小。在这样的情况下可以认为本方法为更高层次的调度提供了一个没有迁移代价的平台。这样在调度时,可以不用考虑迁移带来的代价,为实现简便高效的调度算法提供可靠保障。

表 3　插桩时间和进程状态转换时间

测试程序	首次代码翻译时间(ms)	原始代码上下文到翻译后代码上下文平均转换时间(ms)	翻译后代码上下文到原始代码上下文平均转换时间(ms)
172. gird	165	0.72	0.72
173. applu	175	0.80	0.78
178. galgel	240	0.74	0.74
252. eon	160	0.68	0.66
254. gap	145	0.70	0.72
256. bzip2	160	0.66	0.68
CBLAS	400	0.88	0.86
fft	175	0.74	0.74
fir	165	0.72	0.68
iir	160	0.72	0.70
stream	158	0.64	0.66
平均值	191.18	0.73	0.72

3.4　运行效率

插桩后代码的质量对于进程迁移来说也是十分重要的。为衡量翻译后代码执行效率,本文采用的方法是将测试程序绑定到低功耗核上,测量相对于内核模拟提升的比例,来衡量其执行效率。具体执行效率见图 6。

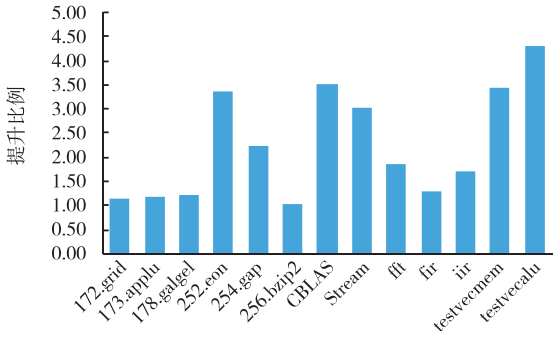


图 6　程序执行效率(根据优化前的执行时间进行归一化)

除了前文提到的测试程序外,这里还额外增加了两个测试程序 testvecmem 和 testvecalu。testvecmem 主要用来测试向量访存指令序列,为大量访存指令的循环;testvecalu 用来测试向量计算指令,为大量向量运算指令的循环。这两个程序一定程度上反映了本文方法相对于内核模拟的加速比的上限,其中向量访存约为 3.43,向量计算约为 4.33。对于以上全部测试集,平均加速比约为 2.25。

4　结　论

在共享指令集异构多核上实现进程迁移,需要从效率、代价、兼容性和可编程性等方面考虑。为了高效、简捷地实现硬件透明的共享指令集异构多核间的进程迁移,本文提出了基于二进制插桩的进程迁移方法。该方法迁移代价低,二进制插桩时间平均为 191ms,状态转换平均时间为 0.73ms,可以适应现在流行的负载平衡间隔和时钟中断间隔,为更高级别的优化提供了可能。同时该方法兼具很好的兼容性,无需对源文件或者编译系统做任何修改。依托龙芯平台,我们实现并验证了基于二进制插桩的异构多核进程迁移方法,作为对比还实现了内核模拟方法,实验结果表明,本方法执行效率平均为内核模拟的 2.25 倍,高效地实现了共享指令集异构多核处理器间的进程迁移,为更高层次的优化提供了保证。

参考文献

[1] Vybrid R Series Automotive Solutions. http://www.freescale.com/webapp/sps/site/taxonomy.jsp? code = VY-

BRID_RSERIES. Freescale 2012

[2] AMD Fusion. http://en. wikipedia. org/wiki/AMD_Fusion. Wiki 2012

[3] Dutta S, Jensen R, Rieckmann A. Viper: A multiprocessor SOC for advanced set-top box and digital Tv systems. *Ieee Design & Test of Computers*, 2001, 18(5): 21

[4] Kahle J, Society I C. The cell processor architecture. In: Proceedings of the 38th Annual IEEE/ACM International Symposiumn on Microarchitecture, 2005. 3-3

[5] 王淼, 王志英, 邬贵明. 一个面向异构多核处理器 Cell 的资源分配模型. 高技术通讯, 2010, 20(12): 1229-1234

[6] Chen J Y, Yang W, Hung T H, et al. A static binary translator for efficient migration of ARM-based applications. In: Proceedings of Workshop on Optimizations for DSP and Embedded Systems, 2008

[7] DeVuyst M, Venkat A, Tullsen D M. Execution migration in a heterogeneous-ISA chip multiprocessor. In: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, London, UK, 2012. 261-272

[8] Li T, Brett P, Knauerhase R, et al. Operating system support for overlapping-ISA heterogeneous multi-core architecture. In: Proceedings of the Sixteenth International Symposium on High-Performance Computer Architecture, 2010. 1-12

[9] Vonbank D G, Shub C M, Sebesta R W. A unified model of pointwise equivalence of procedural computations. *Acm Transactions on Programming Languages and Systems*, 1994, 16(6): 1842

[10] Veldema R, Philippsen M. Near overhead-free heterogeneous thread-migration. In: Proceedings of the IEEE International Conference on Cluster Computing, 2005. 1-10

[11] Smith P, Hutchinson N C. Heterogeneous process migration: the tui system. *Software-Practice & Experience*, 1998, 28(6): 611

A binary-instrumentation based execution migration method for shared ISA heterogeneous multi-core processors

Liu Hongwei[*] [**] [***] [****], Qiu Ji[*] [**] [***] [****], Gao Xiang[*] [**] [****], Chen Yunji[*] [**] [****]

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

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

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

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

Abstract

The characteristics of the excution migration in heterogeneous multi-core processors were analyzed, and a binary-instrumentation based execution migration method for homogeneous multi-core processors was put forward to solve the drawbacks of the present methods for execution migration between shared ISA heterogeneous multi-cores in efficiency, cost, compatibility, or programmability. The proposed migration method based on binary-instrumentation can take full advantage of shared-ISA heterogeneous multi-core to enhance the performance of heterogeneous chip multiprocessors with low cost. And it need not to modify the source code or the compile system. The experimental results obtained from the test on the SPEC procedure showed that its run-time efficiency was 2.25 times of kernel simulation.

Key words: shared instruction set architecture(ISA), heterogeneous multi-core processor, execution migration, binary instrumentation, kernel simulation